Sebastian Böcker

Algorithmic Mass Spectrometry

From molecules to masses and back again

Version 0.9.3 from November 7, 2026

Creative Commons License V4.0: attribution, non-commercial, no derivatives

for Claudia and Frederik



After ten years, I have still not found the time and energy to finish this textbook. Hence, I decided to put it out now, even though it is definitely not "finished". Please help me improve this textbook! Send me any bugs, nonsense, etc. so I can correct it. If you are an expert in the field, please consider contributing text. Many thx in advance! Sebastian.

Preface

> *"Reading, after a certain age, diverts the mind too much from its creative pursuits. Any man who reads too much and uses his own brain too little falls into lazy habits of thinking."* (Albert Einstein)
>
> *"I wrote this book, and the person quoting me here is taking credit for it."* (xkcd)

If, on a journey through life sciences, one leaves the beaten path of analyzing genes and genomes, there is a good chance that one will come into contact with a particular analytical method: Mass spectrometry. This analytical technique serves as a tool for the analysis of proteins, metabolites, glycans, lipids, and sometimes even DNA and RNA. To give an idea, the PubMed database[1] lists 32 690 articles published in 2025 that carry the terms "mass spectrometry", "LC-MS", or "GC-MS" in title or abstract. This increased from 13 519 publications in 2010, with a growth rate of about 6 % per year. Mass spectrometry is a high-throughput technique, and a single instrument can produce gigabytes of data every day. On the other hand, interpreting mass spectra can be involved, and manual interpretation by an expert may require a substantial amount of time.

The textbook Algorithmic Mass Spectrometry focuses on the automated analysis of mass spectrometry data using algorithms, combinatorics, and statistical evaluation. On the application side, the focus is on life science questions such as protein or metabolite identification. As mass spectrometry is rapidly evolving, questions and paradigms for the algorithmic analysis perpetually change. But certain aspects of this analysis remain constant, and certain algorithmic techniques have kept reappearing over the last decades. This textbook tries to provide algorithmic techniques for efficiently solving such questions, and to show how these techniques can be used for, say, peptide *de novo* sequencing.

This textbook (you may also call it "lecture notes") is based on material that I have taught at Bielefeld University and the Friedrich Schiller University Jena. Several people contributed to this textbook in one way or another, including (but not limited to) Andreas Hildebrandt, Anton Pervukhin, Bertram Vogel, Birte Kehr, Florian Rasche, Kai Dührkop, Marcel Martin, Marcus Ludwig, Markus Fleischauer, Markus Fricke, Stefanie Dietrich, Tobias Marschall, Zsuzsanna Lipták, and many students of the courses I have taught. Hannes Röst and Magnus Palmblad contributed comments, improvements, corrections and bug reports on previous versions. The data for Fig. 5.1 was provided by William S. Noble. I thank all of you for your help!

An **important notice:** In October 2026, I got the impression that neither I nor anyone else will ever finish this textbook. At least, no human being. Consequently, I have turned to Claude and all the promises of generative artificial intelligence. Whether a textbook is even needed anymore in 2026, is clearly debatable; but be that as it may. Starting with version 0.9.0, Claude will help me correct grammar and spelling, but also hunt down math errors. In version 0.9.1, we did a very detailed fact check, and Claude generated scripts for all of my little algorithms to ensure that my claims are actually correct — neat! In version 0.9.2, Claude has filled in all the blank sections. I asked Claude to write them as I would write them, and I like it. Where Claude wrote several sentences or more that I did check carefully, the margin says "Claude helped!"; next to a chapter heading, it means that Claude wrote large parts of the chapter. Text written by Claude that I did not check myself is marked by "Claude says" in the margin, but I believe there is none of that right now. Version 0.9.3 is mainly just a change of layout.

Despite several rounds of fact checking and testing by Claude, you may still find bugs, inaccuracies and inconsistencies. If you find them, please send me an email, and I will be happy to correct them!

[1] https://pubmed.ncbi.nlm.nih.gov/

What this textbook is about

This textbook is about "combinatorial and algorithmic questions and solutions related to mass spectrometry, with a slight hint of statistics and machine learning." This would not make a good textbook title, though, so the slightly more catchy title "Algorithmic Mass Spectrometry" made the race. I would have much preferred to call it "Computational Mass Spectrometry", in analogy to the fields of computational geometry or computational algebra. But in view of other computational fields such as computational biology, which comprise more than just algorithmics, combinatorics and machine learning, I have decided to use the more specific title.

In more detail, this textbook is about algorithms and combinatorics solutions that I found clever and elegant. Very often, we need some algorithm to get from a given input to a desired output; but also very often, this algorithm is mostly trivial. I have limited myself to cases where I found the resulting algorithms particularly elegant. Sometimes, the problem is potentially mostly irrelevant in practice (see for example glycan "sequencing" in Chapter 11) but the solution is elegant and instructive, so I nevertheless included it. At the same time, this example demonstrates the extent to which we can reuse ideas and concepts developed in this textbook. My selection is obviously limited by what papers I have written or read; and although I have read several hundred papers in this field, that is still only a small fraction of what has been written. If you know some interesting algorithm for a not-too-shabby application, please let me know; see also Chapter 13.

All methods and algorithms are presented with the following requirements in mind:

Specification. For each method presented in this textbook, we will define input and output as unambiguously as possible. As an example, consider peptide *de novo* sequencing: The input will be an (idealized) tandem mass spectrum in the form of a peak list, plus additional information such as the precursor mass. The output will be a peptide sequence or a peptide sequence with "mass gaps", possibly with some kind of quality measurement. It is not acceptable if a method for peptide sequencing will sometimes return a peptide sequence, sometimes something completely different, and sometimes nothing at all.

Generalizability. The methods presented in this textbook will work for *all* input data, given that the input data stays within the defined borders. This does not mean that the results will in all cases be useful: The "bullshit in, bullshit out" saying still holds true. But if the data is of good quality, we do not want methods that might work for certain input, but fail for the next; in particular, we do not want methods where we cannot even explain why the method worked or failed. An example is a peptide *de novo* sequencing method that will not return any peptide sequence for 90 % of the tandem mass spectra, even if all of them are of excellent quality. The algorithmic mass spectrometry literature is full of "anecdotal" methods that worked for a certain type of input, but the authors cannot prove that they will work for all input; sometimes, the authors openly admit that the method will not work for certain input, which is certainly preferable.

Correctness. For all methods and algorithms presented, we will prove that the method does what it is supposed to do — or at least, we will indicate how it can be proven. Algorithmics is actually a very formal science, where a proof is required, showing that a certain input is transformed into a certain output *in all cases*. If a method ultimately joins, say, five different methods from this textbook as subroutines, then it is not acceptable that any of these subroutines might work for some data, and might fail for others.

Running time. This is an extremely important aspect of algorithmics: How fast will my algorithm process the input? This is usually done in the form of a proven *guarantee*, stating that for any input, the algorithm will not require more than this-and-this time. Only the asymptotic behavior of the running time is reported (big-O notation, Landau symbols): All algorithms tend to be sufficiently fast for small inputs; but what happens if the input gets large?

You might argue that "algorithmic mass spectrometry" is rather a part of cheminformatics than of bioinformatics. But most methods and concepts in algorithmic mass spectrometry have been developed by bioinformaticians who wanted to analyze biomolecules, such as proteins, peptides, lipids, metabolites, glycans, or even DNA and RNA. In fact, one can say that "it all started with peptide sequencing", see Chapter 2.

Considering that some readers might not have a background in combinatorics and algorithmics, I have included some "trivial" algorithms that show how you can transform a recurrence into an algorithm, see for example Algorithms 2.1 and 3.1. This is done to convince the reader that these algorithms are, well, trivial. In the same spirit, I have included algorithms on how to derive optimal and suboptimal solutions from dynamic programming tables, see for example Algorithm 2.2. This should be well known to bioinformatics students, as it is very similar to how you compute pairwise sequence alignments. See also Sec. 15.5.

What this textbook is *not* about

There is an enormous number of topics which are *not* covered in this textbook:

- As noted, this textbook is not about *computational mass spectrometry* in the general meaning. For example, aspects such as signal processing, data storage and retrieval, web technology, user interfaces, or file formats will be ignored. We assume that spectra have been preprocessed, so that all we have to deal with are peak lists, where each peak is described by its mass and intensity, and possibly other peak attributes such as "quality of fit".

- Unfortunately, the lack of signal processing details implies that *quantitative approaches are not covered,* either.

- The selection of problems and solutions presented is not a compilation of the *ten (or so) most important problems in algorithmic mass spectrometry.* I do not even know how to rank problems by application importance. In truth, the selection is extremely biased by my own research interests.

- This textbook is *not about machine learning.* In machine learning (ML), we want to automatically learn to recognize complex patterns, and make decisions based on data. Successful applications of ML in computational mass spectrometry have been reported in the literature, for topics such as predicting "proteotypic" peptides [117, 229], post-processing database searches [167], clustering peaks [42] or mass spectra [345], or predicting tandem mass spectra [1, 2, 378]. I might touch upon one or two such applications, but I will not explain the ML machinery that is doing the work; hence, the description has to stay rather shallow.

- This textbook is *not about rule-based approaches.* Such approaches combine hand-crafted sets of rules that have been collected from the MS literature over years or decades. Such approaches, including several commercial tools, dominated the computational analysis of small molecule mass spectra from the 1960s until about 2010; since then, combinatorial and machine learning approaches have taken over. (Note that for lipids, rule-based approaches are still widely used, see Sec. 10.6.) From an algorithmic point of view, this is rather boring; in addition, one might argue that it has not been very successful.

- This textbook is *not a mass spectrometry textbook.* Rather, we will only look at those MS facts that are necessary to understand the task at hand, sometimes deliberately ignoring the complexity of the subject. Additional MS facts will be introduced only when we need them and, hence, are scattered throughout this textbook. There exists a large number of textbooks on mass spectrometry that the reader can choose from [71, 77, 138, 209, 328, 329, 364].

- This is *not a chemistry textbook*, and I am not a chemist. Sometimes, I will be rather careless when it comes to chemical topics, such as molecular formulas. For example, I will write molecular formulas such as S_{87} or $H_{-2}O_{-1}$ even though quite obviously, no such molecules can exist. But this textbook is about computing things, not about chemistry. Doing such "tricks" will allow us to make calculations simpler and more comprehensible. Financial mathematics has benefited a lot from writing things like $\$-10$ although every school kid knows that "minus ten dollars" cannot exist in reality. In some places, I will even deviate from established definitions; for example, when it comes to "monoisotopic mass".

I am *not a statistician*. I have included Chapters 5 and 6 on statistical significance because I believe these are the bare (or bear) necessities you have to know if you want to do algorithmic mass spectrometry. I have done my best not to write any humbug there. If you happen to be a statistician, please be forgiving when you read these two chapters. 😉

Finally, this textbook is *not a review*. Several review articles are written each year; as the field is quickly developing, this textbook is most certainly not the right place for this. Rather, I will stick with the essentials that keep reappearing over the years in this field, as well as questions and applications which I find instructive or enjoyable. In the best case, both.

After this somewhat lengthy disclaimer, let's get started…

A star (⋆) marks a hard exercise, a large star (★) marks a very hard exercise — which I might not know the solution to.

Contents

1 Introduction to Mass Spectrometry **1**
- 1.1 Atoms, elements, and molecules . . . 1
- 1.2 A tiny primer on biomolecules . . . 3
- 1.3 A short history of mass spectrometry . . . 4
- 1.4 Mass spectrometry in a nutshell . . . 6
  - 1.4.1 Ions vs. molecules . . . 6
  - 1.4.2 Charge states . . . 7
  - 1.4.3 General setup of mass spectrometry instruments . . . 7
  - 1.4.4 Ionization sources . . . 9
  - 1.4.5 Mass analyzers . . . 10
  - 1.4.6 Ion detectors . . . 12
- 1.5 Tandem mass spectrometry . . . 12
- 1.6 Sample preparation and separation . . . 13
  - 1.6.1 Tryptic digestion . . . 13
  - 1.6.2 Separation by chromatography . . . 13
- 1.7 Exercises . . . 14

2 Peptide *De Novo* Sequencing **16**
- 2.1 Introduction and data . . . 16
- 2.2 Formal problem definition . . . 17
- 2.3 Spectrum graphs . . . 21
- 2.4 Dynamic programming for ideal data . . . 23
- 2.5 Getting rid of the unrealistic assumptions . . . 26
  - 2.5.1 Additional peaks . . . 26
  - 2.5.2 General edge-weighted spectrum graphs . . . 29
  - 2.5.3 Missing peaks . . . 29
  - 2.5.4 Prefix mass equals suffix mass . . . 31
- 2.6 Ion series: The abc and xyz of peptide fragmentation . . . 33
- 2.7 Post-translational modifications: Enlarging the alphabet . . . 34
- 2.8 A two-step strategy for *de novo* sequencing . . . 35
- 2.9 Shotgun proteomics: Shoot first, ask later . . . 36
- 2.10 Historical notes and further reading . . . 37
- 2.11 Exercises . . . 38

3 Combinatorics of Weighted Strings **41**
- 3.1 Formal problem definitions . . . 41
- 3.2 Counting compomers and strings . . . 43
- 3.3 Formal proof of the counting lemma . . . 45
- 3.4 Finding witnesses and the decision problem . . . 46
- 3.5 Enumerating strings and compomers . . . 47
- 3.6 The Money Changing problem and the Round Robin algorithm . . . 48
- 3.7 Approximating the number of compomers . . . 52
- 3.8 Historical notes and further reading . . . 53
- 3.9 Exercises . . . 54

4 Comparing Mass Spectra and Database Searching **57**
4.1 Unit mass accuracy: The dot product score . . . 58
4.2 On the eternal beauty of dot products . . . 59
4.3 Mass thresholding: Not an inner product . . . 60
4.4 The probability product kernel: An inner product at last . . . 61
4.5 Matching spectra and the peak counting score for high mass accuracy . . . 64
4.6 Additional and missing peaks: To score or not to score? . . . 66
4.7 Stochastic model for scoring mass deviations . . . 67
4.8 Stochastic model for scoring intensity deviations . . . 69
4.9 Stochastic model for scoring intensities against barcode peaks . . . 71
4.10 Likelihood score for high mass accuracy, both spectra with intensities . . . 71
4.11 Smarter ranking: Beyond the one-size-fits-all score . . . 72
4.12 Hypothesis-driven recalibrating . . . 73
4.13 Historical notes and further reading . . . 77
4.14 Exercises . . . 78

5 Significance: p-values and E-values **80**
5.1 p-values and E-values . . . 81
5.2 A naïve approach for estimating p-values . . . 82
5.3 Parametric distributions . . . 83
5.4 Exact computations using dynamic programming . . . 86
5.5 Calibrated scores and p-value correction . . . 88
5.6 Issues of p-value estimation . . . 89
5.7 Further reading and other approaches . . . 90
5.8 Exercises . . . 91

6 Significance: Decoy Databases and False Discovery Rates **93**
6.1 False discovery rates and q-values . . . 93
6.2 Using random number generators to estimate false discovery rates . . . 94
6.3 Decoy databases and false discovery rate estimation . . . 95
6.4 How to create a decoy database for peptides . . . 97
6.5 Searching separately in target and decoy database . . . 99
6.6 Posterior error probabilities . . . 100
6.7 Evaluation of decoy databases and false discovery rate estimates . . . 101
6.8 The limits of decoy databases: Metaproteomics and metabolomics . . . 103
6.9 Historical notes and further reading . . . 105
6.10 Exercises . . . 106

7 Isotope Distributions and Isotope Patterns **107**
7.1 Isotopes . . . 107
7.2 Isotope distributions and isotope patterns . . . 110
7.3 Simulating isotope patterns . . . 112
7.4 Faster approximation by the fast Fourier transform . . . 116
7.5 Sulfur and other mavericks . . . 119
7.6 Isotope patterns of peptides and proteins . . . 120
7.7 Simulating isotopologues . . . 122
7.8 Formal proof of the folding theorem . . . 122
7.9 Historical notes and further reading . . . 123
7.10 Exercises . . . 125

8 Decomposing Isotope Patterns 129
8.1 Decomposing real numbers . . . 129
8.2 Good choices for the blowup factor . . . 133
8.3 Scoring isotope patterns . . . 134
8.4 Integrating chemical knowledge . . . 136
8.5 On the number of molecular formulas over CHNOPS . . . 137
8.6 Decomposing amino acids . . . 138
8.7 Decomposing average masses . . . 140
8.8 Detour: Smarter rounding . . . 141
8.9 Historical notes and further reading . . . 142
8.10 Exercises . . . 142

9 Fragmentation trees 144
9.1 Naïve approach: Fragmentation bushes . . . 144
9.2 Formal definition of fragmentation trees . . . 144
9.3 Fragmentation graphs and the Maximum Colorful Subtree problem . . . 146
9.4 Exact algorithms for the Maximum Colorful Subtree problem . . . 148
9.4.1 Dynamic programming and fixed-parameter algorithmics . . . 148
9.4.2 Integer linear programming . . . 149
9.5 Heuristic algorithms for the Maximum Colorful Subtree problem . . . 152
9.6 Edge weights in the fragmentation graph . . . 154
9.7 Finding the precursor molecular formula . . . 155
9.8 Historical notes and further reading . . . 155
9.9 Exercises . . . 157

10 Searching metabolite structure databases 159
10.1 More facts about metabolites . . . 159
10.2 Representing molecular structures as graphs . . . 160
10.3 Searching in metabolite structure databases . . . 162
10.4 *De novo* structural elucidation of small molecules . . . 164
10.5 Matching masses to molecular substructures . . . 166
10.6 Lipids . . . 167
10.7 DENDRAL . . . 168
10.8 Historical notes and further reading . . . 168
10.9 Exercises . . . 171

11 Glycan *De Novo* Sequencing 172
11.1 Glycans and glycan topologies . . . 173
11.2 Glycan fragmentation . . . 175
11.3 The candidate generation problem . . . 177
11.4 An exact algorithm for glycan candidate generation . . . 179
11.5 Mostly old wine in new skins . . . 180
11.6 Evaluation of glycan topology candidates . . . 182
11.7 Counting glycan topologies . . . 182
11.8 Historical notes and further reading . . . 186
11.9 Exercises . . . 187

12 Priors and Machine Learning: Overfitting and Self-Fulfilling Prophecies 189
12.1 Priors and prior information . . . 190
12.2 Pirates in the Caribbean . . . 191
12.3 Master of the obvious . . . 191

12.4 Hidden priors . . . 193
12.5 Cross-validation . . . 194
12.6 Data leakage . . . 195
12.7 Cute dogs and smart horses . . . 196
12.8 Independent data . . . 197
12.9 Building a skyscraper from a bag of bricks . . . 198
12.10 Filtering vs. priors and the two-step approach . . . 199
12.11 Further reading . . . 200
12.12 Exercises . . . 201

13 The missing chapters **202**
13.1 Signal processing and peak picking . . . 203

14 Gazing into the Crystal Ball **206**

15 Mathematics and computer science **208**
15.1 Writing numbers . . . 208
15.2 Masses . . . 208
15.3 Floating-point arithmetic and numerical stability . . . 209
15.4 Graph theory . . . 209
15.5 Dynamic programming . . . 210
15.6 Logarithms . . . 211
15.7 Approximations . . . 212
15.8 Computational complexity . . . 213

16 Conclusion **215**

Bibliography **217**

1 Introduction to Mass Spectrometry

"All science is either physics or stamp collecting." (attributed to Ernest Rutherford)

"The more I learn about experiments, the less I believe in them." (Cédric Notredame)

Genomics and transcriptomics have been tremendously successful in the last 40 years. Indispensable prerequisites for this success have been our capability to amplify DNA via polymerase chain reaction and to read its sequence with low error rates. If we shift to proteomics (the large-scale study of proteins), then it becomes evident what the analytical problems are: First, there is no way to amplify proteins. Second, there is no experimental technique to sequence proteins which could compete with classical Sanger sequencing [307], a method for DNA sequencing which is more than 40 years old. Analysis is further complicated by the fact that proteins fold into complex three-dimensional structures, and protein function can only be understood if we take into account these structures. But even if we ignore the latter problem — and we will do so throughout this textbook — it is understood that our situation is much worse than for genomics and DNA analysis. This is even more so the case for metabolite and glycan analysis.

Here, mass spectrometry comes into play. This analytical technique was developed more than a century ago, and has been routinely applied to biomolecules since the advent of "soft" ionization techniques. Mass spectrometry has many advantages, such as high sensitivity when measuring low-concentration molecules, high speed enabling high-throughput experiments, or high accuracy that allows us to determine the mass of a molecule with outstanding precision. But there is one peculiarity of mass spectrometry that makes its analysis quite different from, say, genome sequencing data: We can only derive a single physical property of the molecules or fragments under consideration, namely, their mass (or, more precisely, their mass-to-charge ratio). Computing the mass of a known molecule is trivial. But how can we get back? How can we make claims about the identity of molecules in our sample, when the only information we have available is the mass of the molecule and, possibly, its fragments?

Computational mass spectrometry is a newly emerged field of research in bioinformatics with connections to signal processing, database storage and retrieval, machine learning, sequence analysis, statistics, discrete mathematics and graph theory, computational geometry, and others. Computational mass spectrometry addresses the automated analysis of mass spectrometry data, and is of fundamental importance due to the high-throughput nature of mass spectrometry data.

This chapter is meant as a short introduction, to get things started. It is not meant as a reference that you can come back to in five to ten years, on a particular subject matter. Instead, it introduces the "bare necessities" from chemistry and mass spectrometry, to provide somebody from bioinformatics or computer science with enough information to dive into the application. Many details have been simplified as much as possible, and you might want to look into the literature to learn what is truly going on — in case this is already known. These simplifications are made to set the focus on the computational side. Also, details can get extremely sophisticated, and easily fill a textbook — and as it turns out, many textbooks *have* been filled with these details [71, 77, 138, 209, 328, 329, 364].

1.1 Atoms, elements, and molecules

I now give a very short and somewhat oversimplified introduction to atoms and elements. For the moment, we limit ourselves to the pure essentials that are needed to get ourselves going. For more details, in particular on isotopes and isotope distributions, see Chapter 7.

Atoms are the building blocks of matter that cannot be decomposed chemically. With the exception of very particular environments such as neutron stars, all matter surrounding us is composed of atoms.

element	symbol	mass (Da)
hydrogen	H	1.007825
carbon	C	12.0
nitrogen	N	14.003074
oxygen	O	15.994915
phosphorus	P	30.973762
sulfur	S	31.972071
proton	p^+	1.007276
electron	e^-	0.000549

Table 1.1: Six biologically important elements with monoisotopic masses in Dalton, rounded to six digits. The masses of the electron (and, for comparison, the proton) are required to compute the mass of an ion (i.e., an ionized molecule), see Sec. 1.4.1.

Atoms in turn are usually composed of three types of massive subatomic particles: *electrons* which have a negative charge, *protons* which have a positive charge, and *neutrons* which have no charge. Protons and neutrons make up the atomic nucleus and are called *nucleons*. Atoms have no charge, and must contain the same number of protons and electrons; if this identity is disrupted, the resulting particle is called an *ion*.

Atoms are classified by their *atomic number*, that is, the number of protons in the atom, that defines which *element* the atom is. All atoms with identical atomic number share the same chemical behavior and cannot be differentiated chemically. The elements most abundant in biomolecules are hydrogen (symbol H) with atomic number 1, carbon (C, atomic number 6), nitrogen (N, 7), oxygen (O, 8), phosphorus (P, 15), and sulfur (S, 16). The "backbone" of all biomolecules is made from carbon. Less abundant elements include bromine, fluorine, iodine, silicon, copper, zinc, selenium, and tungsten.

Atoms of the same element can differ in their number of neutrons: Such atoms are called isotopes of the element. Different isotopes occur naturally: For example, carbon can have six or seven neutrons, with relative abundance 98.93 % and 1.07 % in nature, respectively. We will ignore this problem for the moment, and come back to it in Chapter 7. For the moment, we assume that all atoms of each element are *monoisotopic*: This is the natural isotope with lowest mass which, for the elements in Table 1.1, is also the most abundant one; but see Chapter 7.

The *mass* of an atom is measured in "unified atomic mass units" with symbol "u". In biochemistry and molecular biology, the term "Dalton" and the symbol "Da" are used for the same quantity, and we will stick with this notation in the following. In 1961, the International Union of Pure and Applied Chemistry defined the unified atomic mass unit to be 1/12 of the mass of one atom of the carbon-12 isotope.[1] An atom that contains n protons and neutrons will have a mass of roughly n Dalton. This is only a rough estimate, however, since it does not account for the mass contained in the binding energy of an atom's nucleus. This explains the *mass defect*, the difference between the atom's mass and the larger sum of masses of the contained protons, neutrons, and electrons. A word of warning: In mass spectrometry, the term *mass defect* is also used for the difference between the exact mass and the nominal mass of a molecule, which can be positive or negative. See Table 1.1 for the monoisotopic masses of the six elements most abundant in living beings. In this textbook, we will often leave out the unit "Dalton", in particular in the mathematical context of weighted alphabets. *It should be implicitly understood that all masses in this textbook are measured in Dalton, unless explicitly stated otherwise.*

A *molecule* consists of a stable system of two or more atoms. Molecules are the smallest particles that retain the chemical properties of the pure chemical substance containing them. The atoms in a molecule are joined by a chemical bond through shared pairs of electrons. The *chemical formula* reflects the exact

[1] Be warned that until 1961, physicists defined 1 amu (atomic mass unit) as 1/16 the mass of one oxygen-16 atom, whereas chemists used 1/16 of the higher *average* mass of oxygen (the atomic weight) as their unit; see Chapter 7.

number of atoms that compose the molecule. A chemical formula may also supply information about the types and spatial arrangement of bonds in the chemical. We use the term *molecular formula* to indicate that we are solely interested in the number of atoms that compose the molecule. Molecules with the same atoms in different arrangements are called *isomers*. For example, the amino acids leucine and isoleucine are isomers. The *nominal mass* (nucleon number) of a molecule is the sum of protons and neutrons of the constituting atoms. The *mass* of a molecule is the sum of masses of the atoms it is composed of.

A chemical *compound* is somewhat similar to a molecule: Unlike a molecule, it has to be made from more than one element, so that O_2 is a molecule but not a compound. More importantly, compounds can be held together by other than covalent bonds, such as ionic or metallic bonds. Measurement via mass spectrometry and soft ionization leaves (most) covalent bonds intact, but ionic bonds are definitely broken. Hence, it is sometimes ambiguous if we talk about the compound present in the sample, or the molecule(s) measured in mass spectrometry. For understanding this book, you can usually think of the terms "molecule" and "compound" as interchangeable.

1.2 A tiny primer on biomolecules

This section introduces the "players": Computational mass spectrometry, to the largest extent, deals with the analysis of biomolecules. For those who are not familiar with this subject matter, I recapitulate some important facts. Everybody else immediately jumps to the next section.

The "manual of life" is written in deoxyribonucleic acids (DNA): It contains the genetic instructions specifying the biological development of all cellular forms of life.[2] A DNA polymer is a chemically linked chain of nucleotides, each of which consists of a sugar, a phosphate and one of four kinds of bases, namely adenine, cytosine, guanine, and thymine. When encoding information, DNA usually appears in the form of a double strand or double helix. The two strands of a DNA double strand usually form a perfect reverse complement of each other. As a macromolecule, a DNA molecule can have a length of several centimeters. The genome of an organism is, roughly speaking, the total information that is encoded in the DNA of its cells. Every cell of an organism carries a (mostly) identical copy of the genome, with few exceptions such as gametes or mutations.

There exist efficient experimental techniques for analyzing DNA, starting from Polymerase Chain Reaction (PCR) that allows us to replicate DNA at an exponential rate [13], Sanger sequencing [307] that has been used to sequence the human genome and a few others, next-generation sequencing (most prominently by Illumina) that can sequence terabases of DNA per run, to third generation methods for long reads (Pacific Biosciences, Oxford Nanopore). Mass spectrometry never had a dominant role for the analysis of DNA, unlike for proteins; with the advent of second and third generation sequencing, this will be even more so. Apart from a few pioneering methods, to which the author of this textbook has contributed to some extent [25, 99, 210], computational MS does not and will not deal with the analysis of DNA molecules.

Ribonucleic acids (RNA) are biochemically distinguished from DNA by the presence of an additional hydroxyl group, and the use of uracil instead of thymine. A stretch of DNA can be transcribed into RNA, such as messenger RNA (mRNA) encoding proteins. In eukaryotes, certain parts of the RNA molecules are spliced out and the remaining parts are joined, respecting the original order. Through alternative splicing, one DNA sequence can be transcribed into many different mRNA molecules. Other types of RNA exist that do not encode proteins, such as transfer RNA (tRNA) or the famous microRNAs (miRNA) [208]. In the lab, there exist several experimental techniques for analyzing RNA, such as next generation sequencing or microarrays. Regarding RNA analysis and mass spectrometry, the same holds as for DNA and MS.

Finally, the mRNA is translated into a *protein*: Similar to a DNA strand that is a chain of bases, a protein consists of amino acids joined by peptide bonds. An *amino acid* consists of a carboxyl group, an amino group, and the side chain that is specific to each amino acid. Twenty amino acids are encoded by the standard genetic code and are called proteinogenic amino acids. Often, amino acids are modified after

[2] Biology has few rules without exceptions: RNA viruses encode their genetic instructions in RNA.

translation, referred to as Post-Translational Modifications (PTMs). These modifications are not encoded in the DNA or RNA template. The sequence of amino acids constitutes the *primary structure* of the protein. Proteins fold into complex secondary structures (alpha helix, beta sheet) and tertiary structures (spatial relationships in space) that are crucial for their diverse functions. Proteins can also be part of a protein complex, sometimes called quaternary structure. Proteins are essential to the structure and function of all living cells and viruses. Many proteins are enzymes or subunits of enzymes, and catalyze chemical reactions. Other proteins play structural or mechanical roles, are involved in immune response, or the storage and transport of ligands.

Proteins range in size from below 100 amino acids to several thousand amino acids: The muscle protein titin consists of a single amino acid chain of up to about 34 000 residues. Short sequences of amino acids, as well as parts of digested proteins (see below), are referred to as *peptides*. A variety of PTMs exist in protein biosynthesis, such as the formation of disulfide bridges, or attachment of biochemical functional groups by phosphorylation, acetylation, alkylation and methylation, isoprenylation, glycosylation, and others. Presumably the largest PTM is glycosylation, where a (small or large) glycan is attached to one amino acid of the protein, see below.

Metabolites are the intermediates and products of metabolism which, in turn, is the entirety of all chemical reactions that happen in living beings to maintain life. Metabolites are rather small, with mass usually below 1000 Dalton. Examples of metabolites are amino acids, monosaccharides, or adenosine-5'-triphosphate (ATP), the energy currency of the cell. Primary metabolites are directly involved in growth, development, and reproduction of a cell or organism, whereas secondary metabolites are not directly involved in those processes. Most of the secondary metabolites in any given higher eukaryote remain unknown. Unlike for proteins, genome sequencing usually does not allow us to deduce the structure of the metabolites. Also unlike proteins or glycans that are made from smaller monomer building blocks, the molecular structure of metabolites is not restricted. This results in a huge variety and complexity of these molecules. We will come back to metabolites in Chapters 9 and 10.

There are many sub-classes of metabolites that have particular structural restrictions: For example, *lipids* include fats, waxes, sterols, and fat-soluble vitamins. Lipids may be broadly defined as hydrophobic or amphiphilic small molecules. Other sub-classes of metabolites include nucleotides, amino acids, monosaccharides, steroids, or terpenes. With the exception of lipids (Sec. 10.6), we will not further discuss or utilize the peculiarities of these sub-classes.

Glycans are the third major class of biopolymers, and are built from simple sugars (monosaccharides). A large number of monosaccharides exist, but only a few are present in an individual species or cell. Glycans can be assembled in a tree-like structure, making their primary structure considerably more complex than that of proteins. Large glycans include starch, cellulose, and chitin. Glycans can be attached to proteins or lipids, but may also be free molecules. Glycosylation, the attachment of glycans to proteins, is presumably one of the most extensive and complex protein PTMs. Glycans are believed to play an important role in cell growth and development, tumor growth and metastasis, or immune recognition and response. As for metabolites, the structure of glycans cannot be directly inferred from the genome sequence of an organism. We will come back to glycans in Chapter 11.

1.3 A short history of mass spectrometry

Mass spectrometers constitute a large and diverse class of instruments. Their development began at the end of the 19th century with the research on Kanalstrahlen. In 1913, experiments by Sir Joseph J. Thomson gave evidence of the existence of stable (non-radioactive) isotopes. In the early 20th century, Francis W. Aston (who helped to design Thomson's equipment) and Arthur J. Dempster, along with many others, proved the existence of numerous isotopes, and provided measurements of their abundances and masses. In the following years, mass spectrometry transformed from a somewhat "esoteric" technique into routine instruments used in many labs throughout the world. At the end of World War II, mass spectrometry

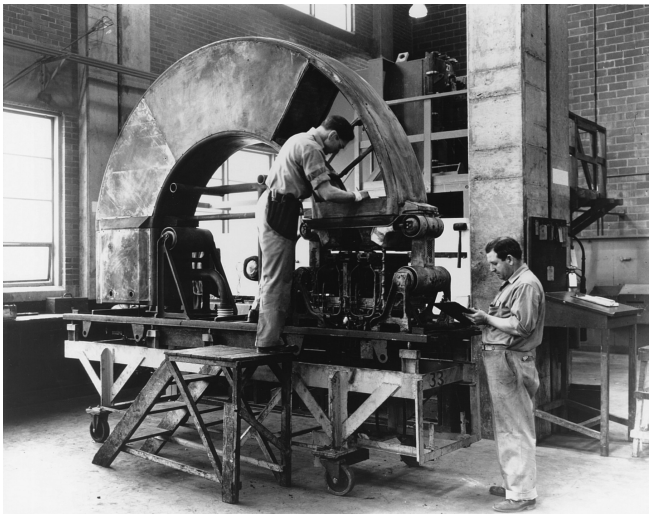


Figure 1.1: The Calutron, used for separating the isotopes of uranium. It was developed by E.O. Lawrence during the Manhattan Project. In 1945, calutrons were used to produce weapons-grade ^{235}U to build the nuclear bomb [266, 371]. After the war, gaseous diffusion technology and (even later) gas centrifuge technology replaced the calutrons. Photograph taken from [371], photo source: Y-12 154.

was used to enrich uranium-235 needed to build the infamous uranium bomb, see Fig. 1.1. The same techniques were later used to provide researchers from physics, chemistry, biology, and medicine with separated isotopes of nearly all the elements of the periodic table. After the war, two important new mass analyzers were invented, namely time-of-flight analyzers by William E. Stephens, and quadrupole analyzers by Wolfgang Paul and Helmut Steinwedel. Also, mass spectrometry was increasingly used to analyze complex organic molecules. Mass spectrometers were coupled to separation techniques such as gas chromatography, pioneered by Roland S. Gohlke and Fred W. McLafferty, a combination which is still frequently in use today. Development continued throughout the following years: Tandem mass spectrometry, developed in 1966, coupled two or more mass spectrometers where one machine served as a source for the next. In the same year, peptide sequencing using mass spectrometry was pioneered by Biemann, Cone, Webster and Arsenault [22]. Even back then, the interpretation of the mass spectrum was supported by a computer program. Fragmenting peptides by Collision Induced Dissociation (CID) was developed in 1968 by Keith R. Jennings and Fred W. McLafferty and is, today, the standard technique for peptide identification. Of particular interest for analyzing proteins and other biomolecules is the development of "soft" ionization techniques, namely Electrospray Ionization (ESI) by John B. Fenn, and Matrix-Assisted Laser Desorption Ionization (MALDI) by Michael Karas and Franz Hillenkamp. In 1974, Melvin B. Comisarow and Alan G. Marshall developed Fourier Transform Ion Cyclotron Resonance (FT-ICR) mass spectrometry, which is known for its high resolution and mass accuracy (see below). The Orbitrap was invented by Alexander Makarov in 2000, based on earlier work by Kenneth Hay Kingdon and R. D. Knight, and shows performance similar to an FT-ICR, but without the need for a superconducting magnet. For an overview, see Table 1.2.

Six Nobel laureates received their prize for discoveries and inventions in the field of mass spectrometry: Joseph J. ("J.J.") Thomson (1906, Physics), Francis W. Aston (1922, Chemistry), Wolfgang Paul (1989, Physics), Hans G. Dehmelt (1989, Physics), John B. Fenn (2002, Chemistry), and Koichi Tanaka (2002, Chemistry). Tanaka was the first to desorb intact proteins by a laser, using a different matrix; only afterwards did Karas and Hillenkamp apply MALDI to proteins. That Karas and Hillenkamp did not share the prize caused quite some controversy.

Claude helped!

to 1900	early mass spectrometry
1913	observation of isotopes using MS
1946	Time-of-Flight MS (TOF)
1953	quadrupole analyzers
1956	Gas Chromatography MS (GC/MS)
1966	chemical ionization
1966	Tandem MS (MS^2), peptide sequencing
1966	metabolomics
1968	Electrospray Ionization (ESI)
1968	Collision Induced Dissociation (CID)
1974	Fourier Transform Ion Cyclotron Resonance (FT-ICR)
1984	Quadrupole/Time-Of-Flight Mass Analyzer (QTOF)
1985	Matrix-Assisted Laser Desorption Ionization (MALDI)
1989	ESI on biomolecules
1993	oligonucleotide ladder sequencing
1999	quantitative proteomics and metabolomics with isotope labels
2000	Orbitrap
2004	Electron Transfer Dissociation (ETD)

Table 1.2: Short and incomplete list of important developments and inventions in the field of mass spectrometry, tailored toward biomolecules.

1.4 Mass spectrometry in a nutshell

To understand what an MS instrument is doing, think of a "parallel scale". This is a bathroom scale that you can step on, with a twist: Instead of weighing just a single person, we can measure the weights of many people in parallel. Think of a giant scale, where a million people can step on in parallel: Our parallel scale then tells us that 5 000 people weighed 64.7 kg, and 12 000 people weighed 77.3 kg. We will come back to this example later.

1.4.1 Ions vs. molecules

To analyze molecules in an MS instrument, these molecules first have to be ionized: *Only ions will be affected by the electromagnetic field of the mass analyzer.* Ionization can happen by "attaching or removing protons or electrons". Depending on the applied electromagnetic field, only positively charged ions (positive mode) or negatively charged ions (negative mode) are recorded. For example, soft ionization results in ions of the form $[M+H]^+$ or $[M-H]^-$ where M is the molecule (its molecular formula or mass, respectively), but only one type is recorded by the instrument.

For convenience, we will often talk about the analyzed molecules or fragments, not their charged ion counterparts. Again, it must be understood that we can easily calculate the mass of a molecule or fragment if we know the mass of the corresponding ion, as well as the ionization type responsible for the charge of the ion. As this is straightforward, we will usually completely ignore this point: You can either add some constant number to all ion masses and then work with the corresponding molecule and fragment masses, or you do it the other way round.

Throughout this textbook, we usually assume that the ion is charged through protonation. There is a subtle problem here: Going from the molecule to its ion, do we "add a proton" to its molecular formula; or, do we "add a hydrogen atom and remove an electron"? This has very subtle consequences for our computations: The mass of a single proton (1.007276467) differs from the mass of a monoisotopic hydrogen ^{1}H minus the rest mass of an electron ($1.007825032 - 0.000548580 = 1.007276452$); furthermore, a single hydrogen has a natural isotope distribution (Chapter 7) whereas a proton does not. The differences between these two approaches are so subtle (mass difference is 0.000000015) that you will hardly ever

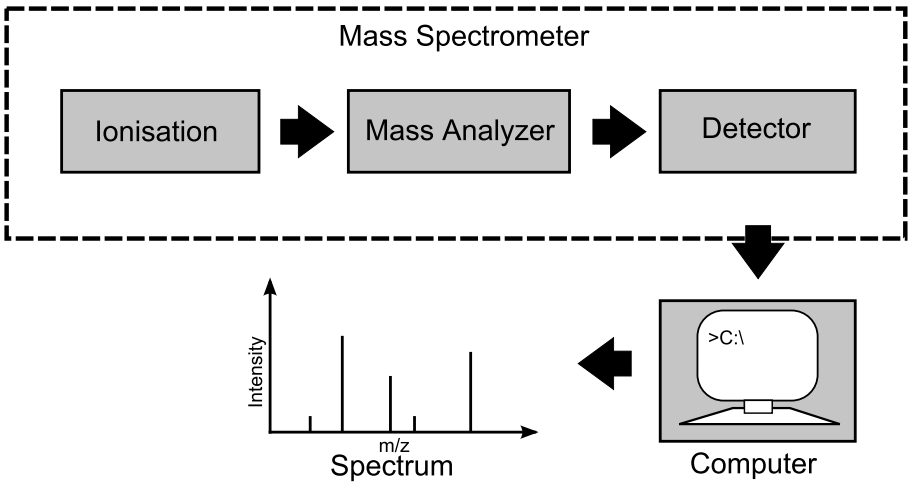


Figure 1.2: Schematic representation of a mass spectrometer.

notice them in practice. Nevertheless, we should keep our computations consistent. After lengthy discussions, me and my group have teamed with the "add H, remove electron mass" side, following suggestions by Ferrer and Thurman [113]. This has the conceptual advantage that we can keep calculations consistent when considering adducts such as $[M+Na]^+$.

1.4.2 Charge states

The charge z of an ion or molecule is a unit-free signed integer $z \in \mathbb{Z}$, where (non-ionized) molecules have charge $z = 0$. MS instruments are unable to distinguish an ion with mass m and charge $z = 1$, from an ion with mass $2m$ and charge $z = 2$ and, more generally, from any ion with mass $z \cdot m$ and charge $z \in \mathbb{N}$. So, we will not be able to record the masses of ions, but only the *mass-to-charge ratio m/z*. It must be understood that in the vast majority of cases, charge state contains little (if any) information about the ion; for this reason, we usually just want to get rid of charges. In many applications and, in particular, for all applications covered in this textbook, we may safely assume that most or all of the ions have a single charge; in case multiply charged ions exist, these are considered annoyances and treated separately, but not considered in the general computational setup. When we talk about recording the masses of ions in our sample, we in fact mean recording the mass-to-charge ratio of the ions, then calculating the masses of the ions under the assumption that, say, all ions are singly charged. There exist applications such as measuring intact proteins where this is not the case, and we have to deal with different and unknown charges in one measurement. In theory, it is not complicated to determine the charge state of a molecule from its isotope pattern (Chapter 7); in practice, it can be more complicated due to overlapping isotope patterns.

1.4.3 General setup of mass spectrometry instruments

In principle, mass spectrometry can be thought of as a three-step process: First, the mixture of molecules that we want to analyze (the *analyte*) has to be ionized in the *ionization source*. Next, analyte ions are separated in the *mass analyzer*. Finally, they hit the *detector* and are recorded. A mass spectrum is then recorded by a computer. Strictly speaking, the computer records a raw signal, and software then decides which signals are reported as peaks, see Sec. 13.1. These three steps are depicted in Fig. 1.2. Note that this separation into three steps is not true for all types of MS instruments: For example, Orbitrap instruments do not separate mass analyzer and detector. But to understand the principles, it is helpful to think of these steps as being separated; for the computational analysis, it is usually not important. It should be noted that most MS instruments operate at very low pressure, close to a vacuum, to minimize the random interaction of analyte ions and other particles inside the instrument.

Let us come back to our example of a parallel scale, and talk about some of the limiting factors of an MS analysis. First, assume that our scale tells us that there was a group of people weighing 77.3 kg; but in reality, these people weigh 77.4 kg. This corresponds to the *mass accuracy* of the measurement. Next, assume that there are two groups of people, one weighing 88.4 kg, the other 88.6 kg. In this case, our scale might wrongly measure only a single group of people with assumed weight 88.5 kg. This corresponds to the *resolution* of the measurement, and gets important if there are several ions with almost identical mass. Next, certain people might be "easier to detect" for our scale than others, so we cannot trust the numbers of people in each group, and only say that "there is a reasonably large group of people with weight 88.5 kg". This corresponds to different "ionization preferences" of different molecules. Finally, if there are not enough people on the scale, we might get no measurement at all, corresponding to the *sensitivity* of the instrument.

Presumably the most important parameter that we have to take into account in our computational analysis is the mass accuracy of the instrument. Mass accuracy is usually measured in "parts per million" (ppm): If we observe a mass (more precisely, mass-to-charge) of 1300 and our mass accuracy is 2 ppm, then the true mass is in the interval 1300 ± 0.0026. Mass accuracy can vastly differ between instruments, particularly between "high resolution" measurements (below 20 ppm) and "unit mass measurements". For the latter, we may assume that we can only measure nominal masses of the ions. From the computational point of view, we need a *guarantee* so that no measured mass ever falls outside the interval; Zubarev and Mann [379] call this the maximum mass deviation, and reserve the term mass accuracy for the statistical spread of mass errors; unfortunately, this is not how all users understand the term, see Chapter 4. The mass accuracy of an instrument is usually provided by the user as a parameter for our computations. Nevertheless, we may analyze the experimental data to see if the given parameter value agrees with the data.

In comparison, the resolution of an instrument is usually much less important: It describes the instrument's capability to resolve two molecules with almost identical mass. For the applications covered in this book, resolution is usually not an issue.[3] It becomes an issue for experimental setups where you cannot separate molecules prior to MS, for example in MS imaging.

The other central parameter of an instrument or instrument type, is much harder to grasp: It is the *sensitivity* of the instrument. What peaks that an instrument should detect, will actually end up in our input list of peak masses? How many false positive peaks will be in there, which do not correspond to any molecule in the sample? Here, I use the term "sensitivity" in the computer science meaning; but the instrument parameter "dynamic range" is also extremely important: A large peak in the spectrum will lead to the non-detection of smaller peaks. If the largest peak has intensity 100 %, does the instrument still record all peaks at 0.1 %? Unfortunately, this parameter is much harder to grasp on the computational side. We should be aware, though, that any experimental mass spectrum has "additional peaks" (peaks that we cannot explain even if we know the "chemical truth") and "missing peaks" (peaks that we expect to see, knowing the "chemical truth"). For the different instrument types, it is usually assumed that Orbitrap and FT-ICR have a smaller dynamic range within a single spectrum, as the number of ions in the trap is limited.

You will sometimes hear people referring to mass spectrometry as "mass spectroscopy". Using this term is not a good idea, as it might lead to confusion with light spectroscopy — and mass spectrometry has nothing to do with light or radiated energy. Similarly, a mass spectrum has nothing to do with a spectrum of light.

I now describe some ionization sources, mass analyzers, and ion detectors in slightly more detail. This description is again vastly incomplete, and rather meant to introduce some important techniques that one encounters when analyzing MS data. See any MS textbook for more details.

[3] In principle, a very high resolution allows us to resolve the isotopologues of a molecule, see Chapter 7; in practice, I am not aware of a computational method that uses these isotopologue patterns and, by doing so, consistently and substantially outperforms computational methods which ignore isotopologues.

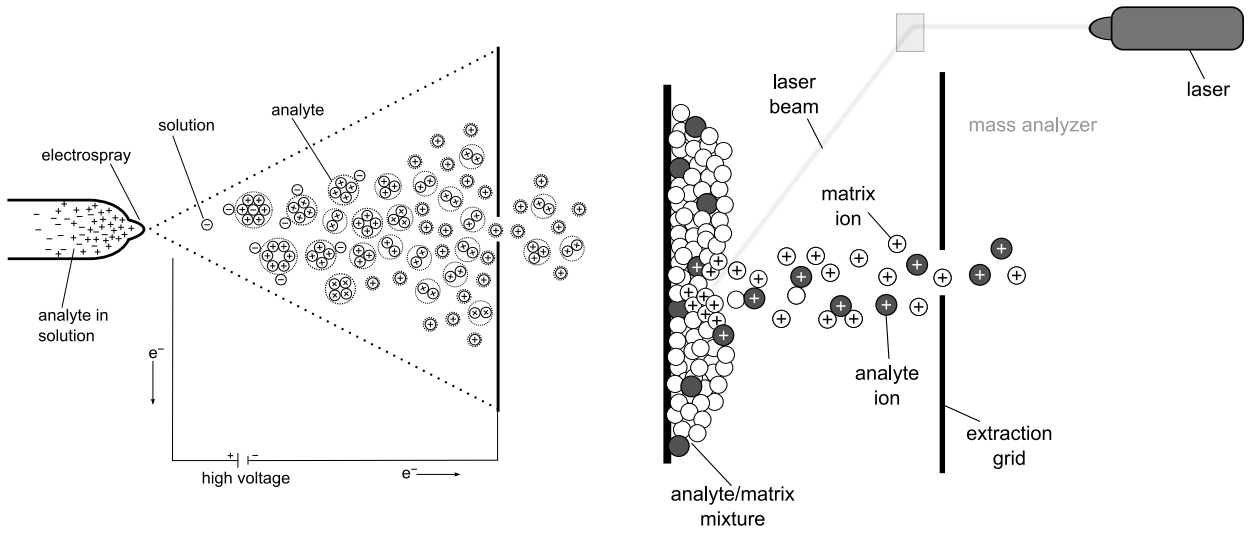


Figure 1.3: Ionization sources: Electrospray ionization (ESI, left) and matrix-assisted laser desorption/ionization (MALDI, right).

1.4.4 Ionization sources

In the ionization source, analyte molecules are converted into ions. Charge can be created by the addition or removal of a proton, or by adding other adduct ions. When analyzing biomolecules, the challenge is to create ions without shattering the analyte molecules: In particular proteins are easily fragmented to uninformative pieces using "hard" ionization techniques.

Electrospray Ionization (ESI) was introduced in 1968 by Malcolm Dole, and developed into a practical technique for mass spectrometry by John B. Fenn in the 1980s. The analyte, dissolved in a large amount of solvent, is pushed through a tiny, highly charged capillary. This liquid pushes itself out of the capillary and forms mist of small droplets. When the solvent evaporates, the excess charges on the droplet surface are forced closer together; they repel each other and eventually break up the droplets. Repeating this process, the droplets get smaller and smaller, until the analyte ions are free of solvent. ESI tends to produce multiply charged ions, in particular for large analyte molecules such as proteins. See Fig. 1.3 (left), and Fenn *et al.* [109] for more details.

Matrix-Assisted Laser Desorption/Ionization (MALDI) was developed in 1985 by Michael Karas and Franz Hillenkamp. The matrix consists of small organic molecules, that absorb energy at the wavelength of the used laser. The method is based on the co-crystallization of the matrix and the analyte components, so that analyte molecules get incorporated into the crystals. The matrix has two functions: It absorbs the light that is fired from the laser, leading to the ionization; and, it protects the molecules of the analyte from being fragmented by the laser. Very large molecules can be ionized by MALDI without fragmenting them. See Fig. 1.3 (right), and Karas and Hillenkamp [173] for more details.

Electron Ionization (EI) was previously called Electron Impact (EI) ionization. It is mainly used in conjunction with gas chromatography (see Sec. 1.6.2) for the analysis of small molecules such as metabolites, see Chapter 10. A beam of energetic electrons is fired at the analyte molecules, inducing ionization and fragmentation. EI is not a "soft" ionization technique, as many of the analyte molecules get fragmented during ionization, often to an extent that no peak is recorded for the mass of the analyte ion. Consequently, EI is practically never used for the analysis of proteins and peptides. But the fragmentation of small molecules is well understood — textbooks have been filled with the details — and the fragmentation spectrum can be used to identify the small molecule by searching in a spectral library.

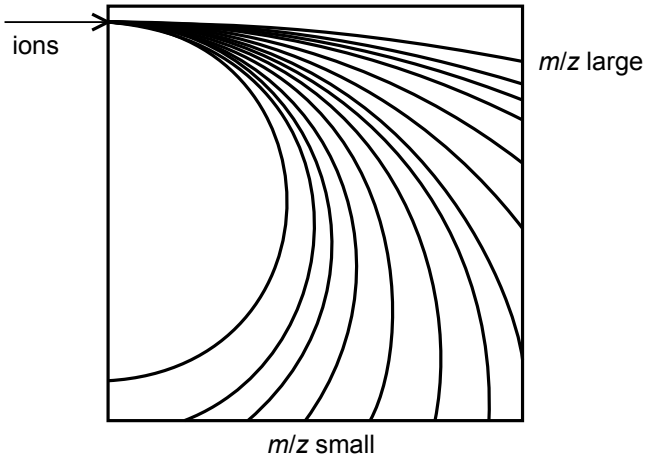


Figure 1.4: Ions in an electromagnetic field.

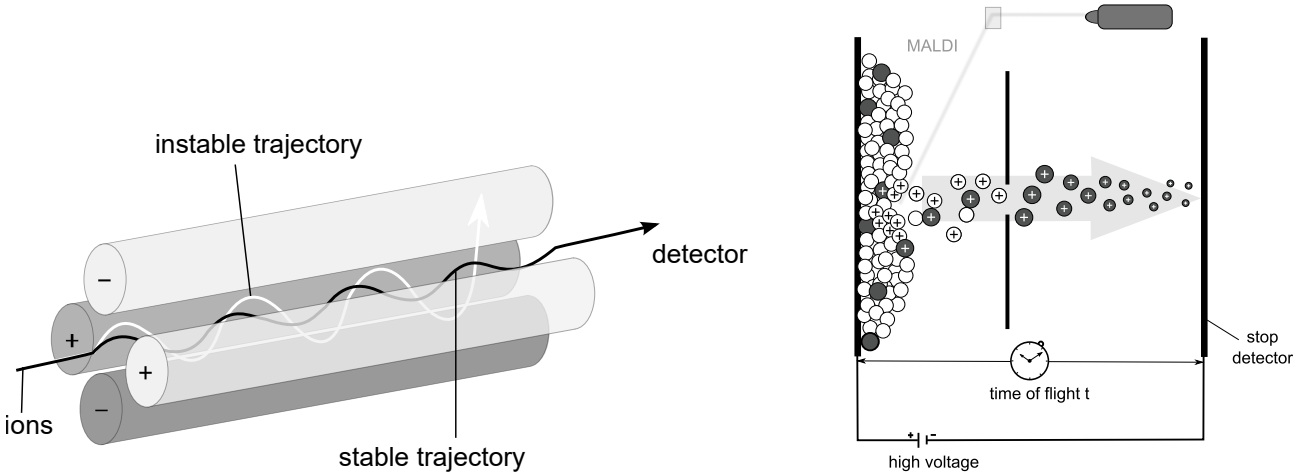


Figure 1.5: Mass analyzers: Quadrupole (left) and Time of Flight (TOF, right).

1.4.5 Mass analyzers

The second step of the mass spectrometry analysis is presumably the most important part, as this determines the accuracy, sensitivity, resolution, and many other aspects of the machine. Many mass analyzers send the accelerated ions through an electric or magnetic field. In such a field, the ions are deflected from their straight line of travel, see Fig. 1.4. The higher the charge of the ion, the larger the force of deflection. On the other hand, ions of small mass are easier to deflect than ions of large mass. The conceptually simplest mass analyzer is the sector field mass analyzer, where ions fly in a curved tube; ions can pass the tube if and only if the mass-to-charge ratio fits the field intensity. By varying the field intensity, we can scan all m/z-values.

In the following, I describe four mass analyzers that are important for the analysis of biomolecules; ion traps and ion mobility devices are also very common.

- The **Quadrupole** mass analyzer consists of four circular and parallel rods, to which oscillating electric fields are applied. The quadrupole is used to filter analyte ions, based on their mass-to-charge ratio. Only ions of a particular mass-to-charge ratio can pass through the quadrupole on a stable trajectory, see Fig. 1.5. By varying the voltages applied to the rods, we scan through the range of mass-to-charge ratios. Quadrupole instruments usually achieve rather low mass accuracy, such as 100 ppm or worse. See Miller and Denton [240] for a detailed overview.

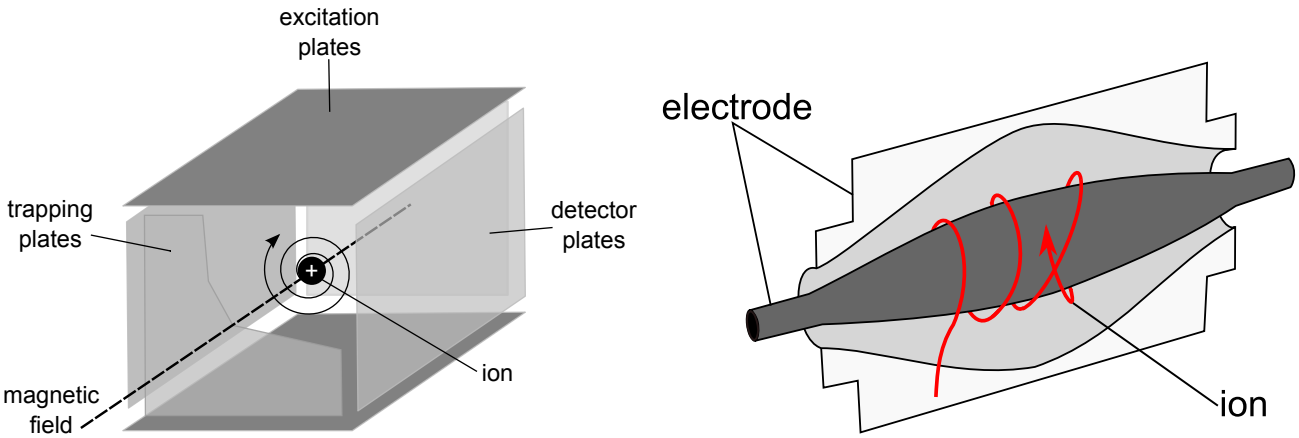


Figure 1.6: Mass analyzers: Fourier Transform Ion Cyclotron Resonance (FT-ICR, left) and Orbitrap (right).

- **Time of Flight (TOF)** first accelerates ions in an electric field so that, in principle, all ions have identical kinetic energy per charge. Then, we measure the time ions need to fly through a field-free drift tube, by sampling the current at the detector at discretized time steps. The time-of-flight of an ion depends on its velocity reached during acceleration in the electric field which, in turn, depends on the mass-to-charge ratio of the ions, see Fig. 1.5. Orthogonal acceleration time-of-flight machines achieve high mass accuracy and resolution. See Guilhaus [140] for a detailed overview.

- **Fourier Transform Ion Cyclotron Resonance (FT-ICR).** These "Penning traps" keep the ions confined in the high magnetic field of a superconducting magnet. The ions circle with frequencies that are inversely proportional to their m/z ratio, see Fig. 1.6. This circling induces an alternating current in the metal plates that make up the trap; this time-varying current can be recorded, so FT-ICR does not require a separate ion detector. The Fourier transform converts this signal over time into a frequency spectrum of the ion motion, which is then converted into a mass spectrum. FT-ICR instruments have outstanding mass accuracy (sometimes below 1 ppm) and very high resolution. FT-ICR instruments measure ions in a non-destructive fashion, something impossible for quadrupole and TOF instruments.

- The **Orbitrap** is an ion trap where moving ions are trapped around an electrode. The electrostatic attraction is compensated by centrifugal force arising from the initial tangential velocity. The special shape of the electrodes also confines the ions axially, where they oscillate along the central electrode, see Fig. 1.6. The crux is not so much the trapping principle itself, which has been known for some time, but rather how to get ions into this trap. The LTQ Orbitrap by Thermo Scientific uses several "tricks" to achieve this. When it was introduced commercially in 2005, the Orbitrap was the first fundamentally new mass analyzer in over 20 years. See Hu *et al.* [156] and Perry *et al.* [271] for more details. Orbitraps also measure ions in a non-destructive fashion.

It must be understood that there is no "perfect" mass analyzer: All come with their particular advantages and disadvantages. For example, quadrupole MS may have a bad mass accuracy, but outstanding sensitivity if we limit ourselves to "few interesting ions" (multiple reaction monitoring).

With regard to mass accuracy, which is presumably the most important parameter next to sensitivity, we may assume QTOF MS reaches a mass accuracy of 10 ppm or better; Orbitrap reaches 5 ppm or better; and FT-ICR measurements can have mass accuracy below 1 ppm. These numbers are only rules of thumb of what one can expect from a "decently modern" instrument of this type in an ordinary lab on an ordinary day, different from the "anecdotal mass accuracy" mentioned in Sec. 4.7.

In the MS literature, there is always a race for "the best" MS instrument; and quite necessarily so, as this drives the development of novel MS methods. But for the computational analysis, we do not get to

choose whether the instrument that generates our data is the spearhead in MS instrumentation. Rather, we have to analyze the data at hand; or, estimate what mass accuracy and other parameters are required to get the biological, biochemical, or chemical answers that we are aiming for. The computational analysis of data from an antiquated instrument is obviously a much harder task and, hence, also a bigger intellectual challenge than analyzing data from a top-flight instrument. At any given time, 99 % of the MS instruments in operation will not be such top-flight instruments; so, there is good reason to develop methods for the other ones, too. In the best case, our computational methods will already work for "low quality" data; but results will hopefully get better for data of better quality.

1.4.6 Ion detectors

Finally, we have to record the ions that were separated in the mass analyzer. Detectors record either the charge induced or the current produced when an ion passes by or hits a surface. The number of ions that leave the mass analyzer for a particular *m/z* value is usually very small, so the signal has to be amplified. Typical ion detectors include electron multipliers, Faraday cups, and microchannel plate detectors. We noted above that for FT-ICR and Orbitrap, the detector is part of the mass analyzer.

It turns out that the actual make of the ion detector is usually not relevant for the computational analysis. Hence, I omit all further details.

1.5 Tandem mass spectrometry

Tandem mass spectrometry describes numerous techniques where ions with a particular mass-to-charge ratio are selected in a first mass analyzer and are introduced into a fragmentation cell. The most prominent fragmentation technique is *Collision Induced Dissociation* (CID), where molecules are passed through a collision cell containing some noble gas, such as helium or argon. As nitrogen is required anyway for the ion source, whereas supplying and refilling an additional gas is expensive, the collision cell is nowadays often filled with nitrogen instead. Fragmentation is achieved by collisions with the neutral atoms of the noble gas, or nitrogen molecules N_2.

Let us take a closer look at the collision. Whereas the picture of flying a space ship into an asteroid field is appealing, it is unfortunately wrong: In fact, the fragmentation is rather a chemical process than a physical one. By colliding with the neutral gas, some of the kinetic energy of the molecule is transferred into internal energy. This energy then triggers a fragmentation pathway which, unfortunately, is usually much more complicated than simply cutting some of the bonds in the molecule.

We call the ion that gets fragmented, the parent ion or *precursor ion*; the ions in the fragmentation spectrum are called daughter ions, product ions or *fragments*. When a singly charged precursor ion is fragmented, the charge of the ion can stay with either of the (usually two) fragments. In this case, the other fragment is not detectable by the MS instrument, and is called a *loss*.[4] It depends on the size of the fragments and, in particular, their molecular structure, which of the fragments is ionized and which is the loss. As we do not fragment a single ion but instead, thousands or millions of identical copies, it is still possible that we can detect both fragments of this fragmentation reaction. In case the fragmented ion is multiply charged, the charges are distributed between its fragments. Again, the distribution of charges to the fragments depends on their size and molecular structure.

In passing, I mention that there exist other fragmentation techniques such as *electron transfer dissociation* (ETD). It must be understood that fragmentation spectra from different fragmentation techniques can look vastly different when fragmenting the same molecule. Recall that Electron Ionization (EI) also

[4] Losses can either be *neutral* or *radical*. Neutral losses have even sum of valences (see Sec. 8.4) whereas radical losses have an unpaired valence electron and odd sum of valences. Similarly, fragments can be radicals; radicals are highly reactive. The majority of losses in tandem MS are neutral losses, but certain radical losses such as $H^{\bullet}$, $O^{\bullet}$, $^{\bullet}OH$, $^{\bullet}CH_3$, $CH_3O^{\bullet}$, $^{\bullet}C_3H_7$, $^{\bullet}C_4H_9$, or $C_6H_5O^{\bullet}$ are "not so uncommon". Clearly, this also depends on the compound class you are fragmenting; for peptides, radical losses are considered rare. On the contrary, radical losses are commonly observed for electron ionization fragmentation.

results in fragmentation of the precursor ion. In comparison to CID and ETD, the fragmentation processes behind EI are much better understood, and their description fills books.

1.6 Sample preparation and separation

Before some molecules can actually be fed to an MS instrument and analyzed there, some steps are taken to make this analysis as simple as possible. In principle, we could directly feed a sample to the instrument, and try to make sense of the data we collect. But this limits the power of our MS analysis, as we have to deal with contamination during the analysis of the data; contaminant signals can superimpose the true signals, dampening or even completely eliminating them. Clearly, it depends on the biological question what we are interested in, what we consider to be "contaminants": In a proteomics experiment, all metabolites are considered contaminants, whereas the converse is true for a metabolomics experiment. As both the experimental setups and the computational analysis of the data for these two fields are very different, it is practically impossible to analyze them in one go. But as we will see below, separation is a crucial step for a comprehensive analysis, so enriching the molecules we are interested in and getting rid of all others is always a good idea.

Furthermore, it is sometimes hard or even impossible to directly analyze the biomolecules at hand: A prominent example is proteins, which are "too large" for MS analysis, at least if we are interested in more than their mass alone. So, proteomics analysis requires us to break the proteins into pieces (peptides) before analyzing them by MS.

I refrain from describing the experimental details that are needed, say, to extract proteins from a cell. In most cases, these are not important for the computational analysis; in fact, I do not know them. In certain situations, though, certain experimental details are relevant for our analysis: An example is "fixed modifications" of amino acids in proteomics experiments (such as the carbamidomethylation of cysteine, see Sec. 2.7) which are due to the experimental setup, not biochemical processes in the sample. Often, we can easily modify the computational approach to take into account such peculiarities, and can safely ignore them when developing our computational methods.

1.6.1 Tryptic digestion

For some time, there appeared to be a competition in the MS community regarding the largest intact protein that could be analyzed by mass spectrometry. Apparently, this competition has come to an end; a possible reason being that the mass of an intact protein does not tell you a lot about the protein. Tandem MS of complete proteins is complicated due to various reasons that are beyond the scope of this textbook [287]. So, the proteomics community came up with a trick: Instead of analyzing a complete protein, one first cleaves the protein into shorter pieces, namely peptides, then analyzes these peptides by (tandem) MS.

Proteins can be cleaved into peptides by chemical or enzymatic methods. To understand enzymatic digestion, note that peptide bonds in proteins are metastable, meaning that they will break spontaneously in the presence of water; but this process is extremely slow. Breaking peptide bonds can be accelerated by proteolytic enzymes such as trypsin, V8, or chymotrypsin. At present, the predominant method for protein cleavage is tryptic digestion: The serine protease trypsin cleaves peptide bonds at the carboxy side of a lysine (K) and arginine (R) residue by hydrolysis, adding a water molecule. This cleavage is inhibited if the lysine or arginine residue is followed by a proline (P). The result of this cleavage is two peptides whose molecular formulas sum up to that of the initial protein, plus H_2O. Unfortunately, digestion is not "perfect", meaning that in some cases, trypsin will miss a cleavage site or cleave at a non-standard site [298].

1.6.2 Separation by chromatography

Mixtures of biomolecules are often too complex to be directly fed into an MS instrument: The abundant analyte ions would make it impossible to detect analyte ions that are less abundant in the sample. For this

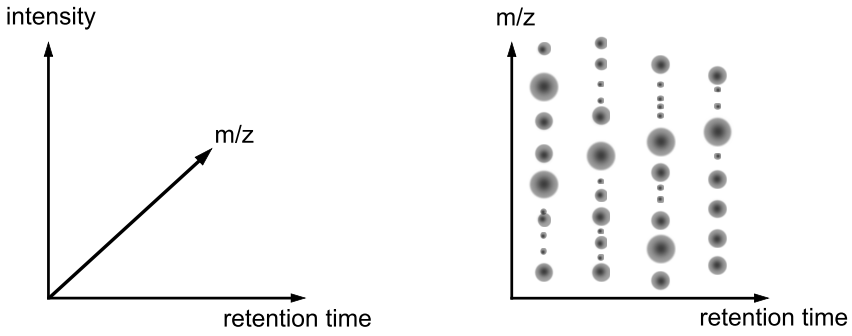


Figure 1.7: A three-dimensional (left) and a two-dimensional (right) LC-MS spectrum.

reason, these mixtures are usually separated before feeding them to the MS instrument. *Chromatography* is the predominant separation technique to be coupled with MS: The analyte molecules are dissolved in a fluid called the *mobile phase*, which carries them through a column holding the *stationary phase*. Different molecules in the analyte travel at different speeds, causing them to separate.

The mobile phase can either be a gas (gas chromatography, GC) or a liquid (liquid chromatography, LC). Since sample molecules have to be heated, gas chromatography is not suitable for larger molecules such as peptides or proteins: These are not volatile, so instead of evaporating, they would decompose in the hot injector and clog the instrument. GC is usually coupled with electron ionization MS for the analysis of small, volatile molecules. In contrast, liquid chromatography is the predominant separation technique for peptides and proteins, but is also increasingly used for the analysis of metabolites. Reversed-phase chromatography has a non-polar stationary phase and an aqueous, moderately polar mobile phase. Here, retention times are shorter for polar molecules and longer for less polar ones.

Each chromatographic peak may contain hundreds of molecules; the LC output is injected into a mass spectrometer. Electrospray ionization (Sec. 1.4.4) is often used for ionization, as it can handle the continuous sample flow. A spectrum is measured at each time step, and the resulting data can be interpreted as a two-dimensional spectrum (Fig. 1.7).

During the LC run, the instrument can select several high peaks from the mass spectrum (termed "MS1" in this context) and acquire individual tandem mass spectra for each of them. Unfortunately, there is not enough time to measure an individual tandem mass spectrum for "each peak" in the MS1; in the extreme, we could suspect a peak at every position of the MS1, and we would have to measure a tandem MS everywhere. Consequently, only a few high-intensity peaks in the MS1 spectrum are selected; we can also instruct the instruments to measure certain peaks or to ignore others. This is called data-dependent acquisition; alternatively, data-independent acquisition fragments all ions in wide *m/z* windows, see Chapter 4.

1.7 Exercises

1.1 Write a program that simulates protein cleavage by tryptic digestion.

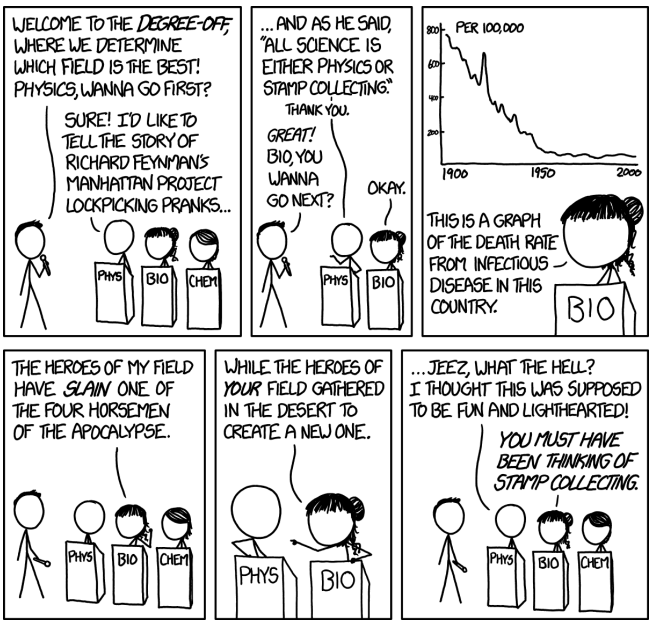


Figure 1.8: Degree-off by xkcd (https://xkcd.com/1520/).

2 Peptide *De Novo* Sequencing

"Everything should be made as simple as possible, but not one bit simpler." (Albert Einstein)

Computers and computer programs have supported mass spectrometry experts in the interpretation of peptide tandem mass spectra since at least the 1960s: For example, Biemann, Cone, Webster and Arsenault [22] used a "computer interpretation" for the sequencing of peptides back in 1966. Numerous such examples can be found in the mass spectrometry literature; they all have in common that this development was not driven by the search for an efficient and general solution of the underlying problem. Rather, programs, algorithms, and methods were developed that analyzed the data at hand; the algorithms and methods themselves never were the objects of investigation.

One can say that the history of computational mass spectrometry started in the years 1999 and 2000: At the Symposium on Discrete Algorithms, Chen, Kao, Tepel, Rush and Church [55] presented a dynamic programming approach for the peptide *de novo* sequencing problem using tandem mass spectrometry. This problem was raised a year earlier by Dančík, Addona, Clauser, Vath and Pevzner [69] at the conference for Research in Computational Molecular Biology, see [70] for the journal version. Some might argue that this history already started back in 1997, when Taylor and Johnson [344] presented the program Lutefisk for the same purpose.[1] Many questions arising in the scope of this analysis, can serve as archetypal questions for computational mass spectrometry.

Before the advent of mass spectrometry, proteins and peptides were sequenced using Edman degradation [96], developed by Pehr Edman in 1950. Amino acids are read step-by-step from the N-terminus of the protein, then cleaved off. The method has certain shortcomings [334] and, in comparison with mass spectrometry, it is very slow and work-intensive.

If you think that the task of peptide sequencing is a sensible thing to do, then you might want to skip this paragraph. But some students might argue that sequencing peptides or proteins is a rather futile task in the time of genome sequencing, since we can infer protein sequences from the genome of an organism. This is a plausible argument, but it is wrong: We just mention a few counter-arguments. In eukaryotes, a single gene can correspond to tens of thousands of proteins due to alternative splicing. Most proteins are edited and modified after translation, and there exists a huge variety of post-translational modifications for this purpose. Certain peptides are not even encoded in the genome, for example the antibiotic Actinomycin D. Not every species is sequenced, not every gene and splice form is annotated. Proteogenomics, an emerging scientific field at the intersection of proteomics and genomics, uses proteomic information from mass spectrometry to improve gene annotations. Finally, tag-based approaches will sequence a short snippet (substring with 3–6 characters) of the sequence before database searching, to improve running time [341]. This list is most likely incomplete, but it is sufficient to make the point.

And why are we sequencing peptides and not proteins? As we noted above, there is a simple answer to this question: If we *could* sequence proteins, then we *would* sequence proteins; but we *cannot*.

2.1 Introduction and data

Tandem mass spectrometry, as introduced in Sec. 1.5, can be used to determine the amino acid sequence of an unknown peptide: A first mass analyzer separates one peptide from many entering the MS instrument. In the fragmentation cell, peptide ions collide with neutral gas molecules, causing them to fragment by

[1] Even others might argue that this history started with DENDRAL back in 1965 [206, 207], see Sec. 10.7; but I would object, as DENDRAL projects did not care about specification, generalizability, correctness, or running time of the developed algorithms.

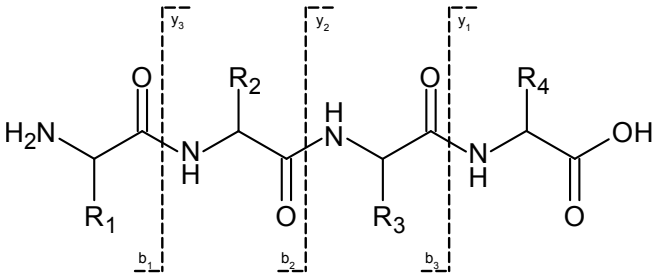


Figure 2.1: Fragmentation of a peptide into b and y ions.

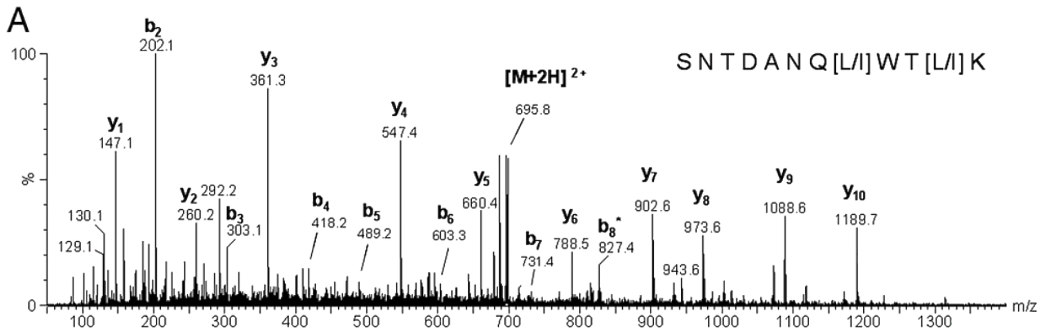


Figure 2.2: Expert-annotated tandem mass spectrum of the tryptic peptide SNTDANQ[L/I]WT[L/I]K originating from a type II ribosome-inactivating protein isolated from *Ximenia americana*. Figure taken from Seidler *et al.* [318].

collision-induced dissociation. A second mass analyzer records the masses of the product ions corresponding to peptide fragments. For *de novo* sequencing, our task is to reconstruct the amino acid sequence solely from this tandem mass spectrum.

See Fig. 2.1 on how peptides fragment. Many years ago, a nomenclature was introduced in the MS community to name the ions commonly resulting from peptide fragmentation. The most common and informative ions are generated by fragmenting the amide bond between amino acids. Resulting ions are called *b ions* or *y ions*: For b ions, the charge is retained by the amino-terminal part of the peptide; for y ions, the charge is retained by the carboxy-terminal part. In subscript, we can indicate the number of amino acid residues in the fragment.

See Fig. 2.2 for a tandem mass spectrum of a peptide that was hand-annotated by an expert. Our task in this chapter is quite simple to describe: Derive an automated method that, given a tandem mass spectrum of a peptide, annotates the spectrum and recovers the underlying peptide sequence. The idea is that we do so solely based on the tandem mass spectrum, without access to databases for, say, protein sequences.

2.2 Formal problem definition

We want to formalize the peptide *de novo* sequencing problem so that we can attack it by combinatorial and algorithmic means. We start with an oversimplified, idealized version of the problem that cannot be applied to experimental data. We first find an algorithmic solution for the simple problem; only then do we show in Sec. 2.5 how to get rid of our simplifying assumptions.

First, we recall some well-known definitions from computer science: A *string* s over the alphabet Σ, denoted $s \in \Sigma^*$, is a sequence of characters $s = s_1 s_2 \dots s_l$ with $s_i \in \Sigma$ for all $i = 1, \dots, l$. Let $|s| := l$ denote the *length* of s. The unique string of length zero is called the *empty* string and denoted ϵ. We write $s = ab$ to indicate that we can concatenate strings a and b to get s. Any string a with $s = ab$ is called a *prefix* of s, any such string b is called a *suffix* of s. If $s = abc$ holds for strings a, b, c then b is called a

symb.	TLC	amino acid	molecular formula	mass (Da)
A	Ala	alanine	C_3H_5NO	71.037114
C	Cys	cysteine	C_3H_5NOS	103.009185
D	Asp	aspartic acid	$C_4H_5NO_3$	115.026943
E	Glu	glutamic acid	$C_5H_7NO_3$	129.042593
F	Phe	phenylalanine	C_9H_9NO	147.068414
G	Gly	glycine	C_2H_3NO	57.021464
H	His	histidine	$C_6H_7N_3O$	137.058912
I	Ile	isoleucine	$C_6H_{11}NO$	113.084064
K	Lys	lysine	$C_6H_{12}N_2O$	128.094963
L	Leu	leucine	$C_6H_{11}NO$	113.084064
M	Met	methionine	C_5H_9NOS	131.040485
N	Asn	asparagine	$C_4H_6N_2O_2$	114.042927
P	Pro	proline	C_5H_7NO	97.052764
Q	Gln	glutamine	$C_5H_8N_2O_2$	128.058578
R	Arg	arginine	$C_6H_{12}N_4O$	156.101111
S	Ser	serine	$C_3H_5NO_2$	87.032028
T	Thr	threonine	$C_4H_7NO_2$	101.047678
V	Val	valine	C_5H_9NO	99.068414
W	Trp	tryptophan	$C_{11}H_{10}N_2O$	186.079313
Y	Tyr	tyrosine	$C_9H_9NO_2$	163.063329

Table 2.1: Proteinogenic amino acids with symbol, three-letter code (TLC), molecular formula of the residue, and monoisotopic mass of the residue. To obtain the molecular formula of the corresponding amino acid, simply add H_2O; to calculate its mass, add 18.010565 Da. Note that isoleucine and leucine are isomers with identical molecular formula. Note also that lysine and glutamine have a small mass difference of only 0.036385 Da.

substring of s. Deliberately, we did not exclude empty strings from these definitions: We say that a string s' is a *proper* prefix (suffix, substring) of s if s' is a prefix (suffix, substring) of s, but neither s nor the empty string, $s' \notin \{\epsilon, s\}$. For the string $s = s_1 s_2 \dots s_l$, we will denote the substring from position i to position j by $s[i \dots j] = s_i s_{i+1} \dots s_{j-1} s_j$.

The first thing we need is an alphabet Σ that our strings are made up of. Analyzing proteins, an obvious choice for this alphabet is the set of all amino acid one-letter symbols,

$$\Sigma := \{\mathsf{A}, \mathsf{C}, \mathsf{D}, \mathsf{E}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{I}, \mathsf{K}, \mathsf{L}, \mathsf{M}, \mathsf{N}, \mathsf{P}, \mathsf{Q}, \mathsf{R}, \mathsf{S}, \mathsf{T}, \mathsf{V}, \mathsf{W}, \mathsf{Y}\}.$$

But this is neither the only possible choice, nor the most reasonable for our application: Regarding the latter point, we note that leucine (L) and isoleucine (I) have exactly the same molecular formula and, hence, also identical mass. Consequently, we will not be able to tell these two apart using mass spectrometry, and we will treat them as a single letter (usually denoted L). On the other hand, we may include the methylated form of certain amino acids, such as methylated arginine (R^*). We will come back to this issue in Sec. 2.7. For the remainder of this chapter, it is sufficient to think of Σ as an arbitrary but fixed set of characters.

For computational mass spectrometry, we have to determine the masses of molecules; for analyzing peptides, we have to know the masses of the characters in our amino acid alphabet. Formally, we assume that a *mass function* $\mu : \Sigma \to \mathbb{R}_{>0}$ is given. To simplify the presentation, we assume that all characters of the alphabet have different masses, so $\mu(z) \neq \mu(z')$ for $z \neq z'$. This is not a real restriction: We can replace characters with identical mass by some artificial character, and at a later stage, we replace this artificial character by any of the original characters.

What are masses $\mu(z)$ in practice? When an amino acid is added to a peptide, a water molecule H_2O is released as the peptide bond is formed. (Chemically speaking, a peptide consists of n amino acids minus

$n-1$ water molecules.) For this reason, we do not report masses of amino acids, but rather amino acid *residues*. To calculate the molecular formula or mass of a peptide, one has to add up the molecular formulas or masses of the constituting amino acid residues, and add H_2O or 18.010565 Da. For example, the peptide ESI has molecular formula

$$C_5H_7NO_3 + C_3H_5NO_2 + C_6H_{11}NO + H_2O = C_{14}H_{25}N_3O_7$$

and mass

$$129.042593 + 87.032028 + 113.084064 + 18.010565 = 347.169250$$

Dalton. See Table 2.1 for the molecular formulas and masses of amino acid residues; we defer further details to Chapter 7.

For the moment, we will deliberately ignore the additional water molecule which has to be added to the molecular formula of the peptide: As we will see below, the true situation is slightly more complicated but, nonetheless, requires only minor modifications to our model. This allows us to define the *mass* of a string $s = s_1 \dots s_n$ over Σ as $\mu(s) := \sum_{j=1}^{n} \mu(s_j)$.

In the previous section, we have seen that tandem MS allows us to measure the masses of both N-terminal fragments (b ions) and C-terminal fragments (y ions) of the unknown peptide s. Computationally speaking, N-terminal fragments correspond to prefixes of s, whereas C-terminal fragments correspond to suffixes of the peptide. To this end, we define the *fragmentation spectrum* $\mathcal{M}(s)$ of a string $s \in \Sigma^*$ as the set of masses of all prefixes and suffixes of s:

$$\mathcal{M}(s) := \{\mu(a), \mu(b) \,:\, a \text{ is prefix of } s,\, b \text{ is suffix of } s\} \tag{2.1}$$

Sometimes, we will refer to $\mathcal{M}(s)$ as the *ideal fragmentation spectrum*, to distinguish between this and the measured fragmentation spectrum. We will call the elements $m \in \mathcal{M}$ either *masses* or *peaks*, depending on the context. The *precursor mass* M of a string s is simply its mass, $M = \mu(s)$. We may assume that we know the precursor mass of the unknown peptide, as this is the mass that we filtered for in the first mass analyzer.

We now present a first example that we will repeatedly use throughout this chapter. As the masses of amino acids are rather "unwieldy", we use an artificial alphabet with much smaller integer masses, so that all calculations can be carried out using pen and paper. The masses from Table 2.1 should be seen as a gentle reminder of what the problem will look like for real-world data.

Example 2.1. Consider the alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ with mass function $\mu(\mathsf{a}) = 2$, $\mu(\mathsf{b}) = 3$, $\mu(\mathsf{c}) = 7$, and $\mu(\mathsf{d}) = 10$. The string $s = \mathsf{acab}$ has prefixes acab, aca, ac, a, and ϵ with masses 14, 11, 9, 2, and 0; and suffixes ϵ, b, ab, cab, acab with masses 0, 3, 5, 12, and 14. This corresponds to the fragmentation spectrum

$$\mathcal{M} := \mathcal{M}(s) = \{0, 2, 3, 5, 9, 11, 12, 14\}.$$

See Fig. 2.3 for the corresponding "mass spectrum". Note that for this example, all proper prefixes and suffixes have distinct masses.

Now, we can formally define the computational problem we are interested in:

Peptide De Novo Sequencing problem. Given a set $\mathcal{M}$ of masses, find a string $s \in \Sigma^*$ such that $\mathcal{M}(s) = \mathcal{M}$, or decide that no such string s exists.

It is important to understand that the challenging part of this problem, is the simultaneous presence of both prefix peaks and suffix peaks. If only prefix peaks were present, then it is easy to solve the problem even in the presence of additional peaks, see Exercise 2.8. The same holds true if only suffix peaks were present, see Exercise 2.1. Finally, the problem is simple if we know, for each peak, whether it is a prefix or a suffix peak. So, our task can also be described as assigning, to each peak, a label "prefix" or "suffix".

To make it easier for us to come up with an algorithm for the problem, we have made or will make several simplifying assumptions. We will show in Section 2.5 how to get rid of all of these assumptions. But for the moment, the assumptions help us to see the core of the problem, without being distracted by "too many details".

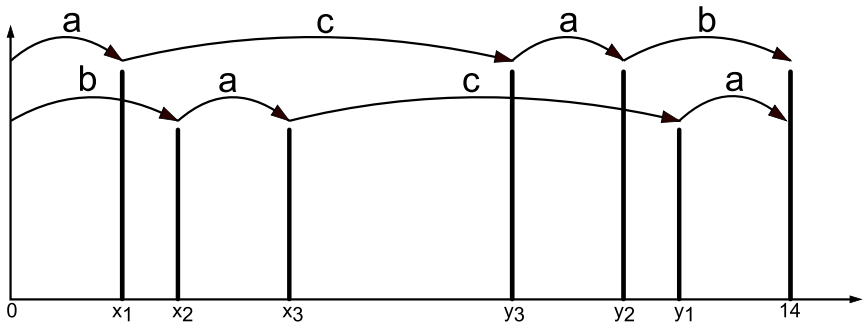


Figure 2.3: Fragmentation spectrum $\mathcal{M}(s)$ for $s = \mathtt{acab}$ from Example 2.1 with prefix peaks (red), suffix peaks (blue), and precursor peak (black).

1. Besides the masses of the prefixes and suffixes of s, no other mass signals are recorded by the instrument. In reality, we usually have to deal with additional peaks that cannot be explained from peptide s, such as chemical noise; other peaks stem from fragmentation events not captured by our simple fragmentation model, or are truly noise in the ion detector. See Sec. 2.5.1.

2. All peaks of the fragmentation spectrum are recorded by the MS instrument, and none are missing. In reality, many peaks that should be present are not detected in the measurement, as they were "lost in the noise": Certain fragmentation events happen too rarely to record the corresponding ions; also, ionization preferences may lead to uncharged fragments that are not detectable by MS. See Sec. 2.5.3.

3. Prefixes and suffixes have different masses: for every proper prefix a and every proper suffix b of s we have $\mu(a) \neq \mu(b)$. In reality, this assumption is less restrictive than it may appear. But the idea behind this simplifying assumption is fundamental for our computational approach.

4. The mass of any fragment is simply the total mass of the constituent amino acid residues. In reality, masses have to be modified with respect to the ion series a fragment stems from, and mass modifications are different for the different ion series, see Sec. 2.6.

5. The MS instrument records exact masses of peptide fragments. In reality, measured masses will deviate from these theoretical and exact masses. This appears to be a simple problem — check if $|m - m'|$ is sufficiently small — but it comes with certain pitfalls, see for example Exercise 2.18. We will come back to this problem repeatedly throughout this textbook.

All of these assumptions are quite natural, except for one: To keep things simple, we initially do not want to think about "ion series mass modification", or the insufficiency of mass spectrometry to record what it should record. But why Assumption 3? This is a somewhat strange assumption, as it artificially limits the space of peptides that we can apply our method to, in contrast to our aspiration for generalizability. Only when we come to the optimization version of our algorithm can we explain why this assumption makes sense, and how we can drop it while simultaneously avoiding the resulting pitfalls. This will be discussed in Sec. 2.5.4.

We now collect several observations regarding our idealized model of fragmentation spectra.

1. Consider a string $s = s_1 \dots s_n$ and its inverse $s^{-1} = s_n s_{n-1} \dots s_1$, then $\mathcal{M}(s) = \mathcal{M}(s^{-1})$. So, a string and its inverse string cannot be told apart using their fragmentation spectra.

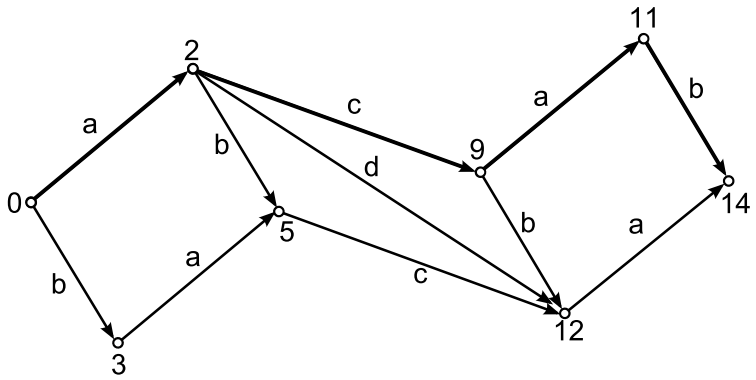


Figure 2.4: Spectrum graph for Example 2.1. Bold edges show the valid path corresponding to string s = acab.

2. Let $\mathcal{M} := \mathcal{M}(s)$ be the fragmentation spectrum of some string s. Then, for each $x \in \mathcal{M}$ we also have $M - x \in \mathcal{M}$.

3. In view of Assumption 3 we have $\frac{M}{2} \notin \mathcal{M}(s)$, as this would result in a prefix and suffix of identical mass.

4. A non-empty string s generates exactly $2|s|$ masses in $\mathcal{M}(s)$: The string s has $|s| - 1$ proper prefixes and $|s| - 1$ proper suffixes, plus masses 0 for the empty string and M for the complete string.

We have to differentiate between observations that only hold for our idealized model, and those that will also hold in practice. It turns out that none of these observations holds for real-world data. Still and all, Observations 2 and 4 will help us to come up with an algorithm for the idealized problem and, later, also for peptide *de novo* sequencing in practice.

2.3 Spectrum graphs

We are given a set of masses $\mathcal{M}$; our task is to solve the Peptide De Novo Sequencing problem by finding a string s with $\mathcal{M}(s) = \mathcal{M}$. To this end, we introduce a novel data structure, called spectrum graph, that allows us to process the data in the spectrum $\mathcal{M}$ more readily. Before we start, let us recall some basic definitions from computational graph theory, see also Sec. 15.4.

A *directed graph* $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subseteq V \times V$. We say that $e = (u, v) \in E$ is an edge from u to v, and we write $e = uv$ for short. A *path* in G is a sequence $p = u_0 u_1 \dots u_l$ of nodes of G, such that $u_{i-1}u_i$ is an edge of G for all $i = 1, \dots, l$.[2] We say that p is a path from $u = u_0$ to $v = u_l$. Let $|p| := l$ denote the *length* of p. A path is called *trivial* if it has length zero, and *non-trivial* otherwise.

A *cycle* in a directed graph $G = (V, E)$ is a non-trivial path from v to v, for some node $v \in V$. A directed graph is *acyclic* if it does not contain any cycles. Informally speaking, "acyclic" means that we cannot walk away from some node v of the graph along directed edges, and ultimately end up in v again. A directed, acyclic graph is called a *DAG*.

After we have introduced the necessary prerequisites from graph theory, let us come back to the *de novo* sequencing problem. Given a set of masses $\mathcal{M}$, the corresponding *spectrum graph* $G := G(\mathcal{M})$ is a DAG $G = (V, E)$ with node set $V := \mathcal{M}$, and there is an edge uv for $u, v \in V$ if and only if there exists some $z \in \Sigma$ such that $u + \mu(z) = v$. We say that edge uv is *labeled* by character z. For each node u, all edges leaving u are labeled differently. It is easy to check that the spectrum graph is acyclic; in fact, its nodes are

[2] Some authors in graph theory call this a "walk", whereas "path" is then reserved for walks where all nodes are distinct — except possibly the first and last node.

ordered, and an edge from node u to v can only exist if $u < v$. The spectrum graph for the mass spectrum $\mathcal{M}$ from Example 2.1 is shown in Fig. 2.4. Note that $M := \max \mathcal{M}$ is the precursor mass of the unknown string. Node 0 is a *source* of the graph as it has no incoming edges; node M is a *sink* of the graph as it has no outgoing edges.

Assuming ideal data, we first use Observation 4: In case $|\mathcal{M}|$ is odd, we can immediately reject the instance. Otherwise, choose n such that $|\mathcal{M}| = 2n + 2$. Then, we can name the masses in $V = \mathcal{M}$ as $\mathcal{M} = \{x_0, x_1, \ldots, x_{n-1}, x_n, y_n, y_{n-1}, \ldots, y_1, y_0\}$ with $x_0 = 0$, $y_0 = M$, and

$$x_0 < x_1 < \cdots < x_{n-1} < x_n < y_n < y_{n-1} < \cdots < y_1 < y_0. \tag{2.2}$$

By Observation 2, we infer that $x_j + y_j = M$ holds for all $j = 1, \ldots, n$; otherwise, we can again reject the instance. From the application standpoint, we note that for every prefix fragment (b ion) we can find the complementing suffix fragment (y ion), and these two add up to the precursor mass. We also infer that the length of the string s that we want to reconstruct, is $|s| = n + 1$.

Any path in the spectrum graph corresponds to a unique string, constructed by concatenating the edge labels of the edges that we visit along the path. In particular, a path from source 0 to sink M corresponds to a string of mass M. Assume that we know the correct string s with $\mathcal{M}(s) = \mathcal{M}$. Then, this string describes a path $v_0 v_1 \ldots v_l$ through the spectrum graph $G(\mathcal{M})$ from 0 to M, where $l = |s|$: From node v_{j-1} we follow the edge labeled s_j to v_j, for all $j = 1, \ldots, l$.

So, in order to recover the string s, it seems reasonable to search for paths from 0 to M in the spectrum graph $G(\mathcal{M})$. One can easily check that for any such path p and corresponding string s, all prefix masses and suffix masses of s are elements of $\mathcal{M}$. Clearly, there may be many such paths: For Example 2.1, we find five paths, namely $0, 2, 5, 12, 14$; $0, 2, 9, 11, 14$; $0, 2, 9, 12, 14$; $0, 2, 12, 14$; and $0, 3, 5, 12, 14$. But for certain paths, the corresponding string may violate our simplifying assumptions:

- The path $0, 2, 5, 12, 14$ corresponds to the string abca; but this string has prefix a and suffix a which obviously have identical mass, violating Assumption 3. This corresponds to visiting both the nodes 2 and $14 - 2 = 12$ in our path.
- The path $0, 2, 12, 14$ corresponds to the string ada; but this string has no prefix or suffix of mass 3, violating Assumption 1.

We want to formalize these two observations: According to Assumption 3, a valid path in the spectrum graph $G(\mathcal{M})$ must visit either $m \in \mathcal{M}$ or $M - m \in \mathcal{M}$, but not both; and according to Assumption 1, it must visit at least one of $m \in \mathcal{M}$ and $M - m \in \mathcal{M}$. Consequently, we say that a path in the spectrum graph $G = (V, E)$ is *valid* if it starts in 0 and ends in M, and for $V = \{x_0, \ldots, x_n, y_n, \ldots, y_0\}$ from (2.2), the path visits *exactly one* of the two nodes x_i, y_i for every $i = 1, \ldots, n$.

Lemma 2.1. *Given a set of masses $\mathcal{M}$ with spectrum graph $G := G(\mathcal{M})$. Let p be a path in G with corresponding string s. Then, $\mathcal{M}(s) = \mathcal{M}$ if and only if p is valid.*

Proof. We have seen above that $\mathcal{M}(s) = \mathcal{M}$ implies that p must be a valid path. We concentrate on the other direction of the proof.

So, let p be a valid path in G, and let s be the corresponding string. We have to show that $\mathcal{M}(s) = \mathcal{M}$ holds. Assume that $p = v_0 v_1 \ldots v_{n+1}$ with $v_0 = 0$ and $v_{n+1} = M$. One can easily check that the prefix a of s of length $|a| = j$ has mass v_j, and that the suffix b of s of length $n+1-j$ has mass $M - v_j$, for all $j = 0, \ldots, n+1$. In view of the definition of $\mathcal{M}(s)$, this is sufficient to show $\mathcal{M}(s) = \mathcal{M}$. □

So, we have transformed the problem of finding the peptide string, into the problem of finding a valid path in the spectrum graph. This new problem is neither simpler nor more complicated than the original one; in fact, both problems are only two sides of the same coin. But as we will see, the graph-theoretical formulation makes it somewhat easier for us to come up with an efficient algorithm for its solution.

Without going into the details, we note that finding valid paths is a particular instance of the Antisymmetric Longest Path problem. For general DAGs, this is an NP-hard problem, see Sec. 15.8. This

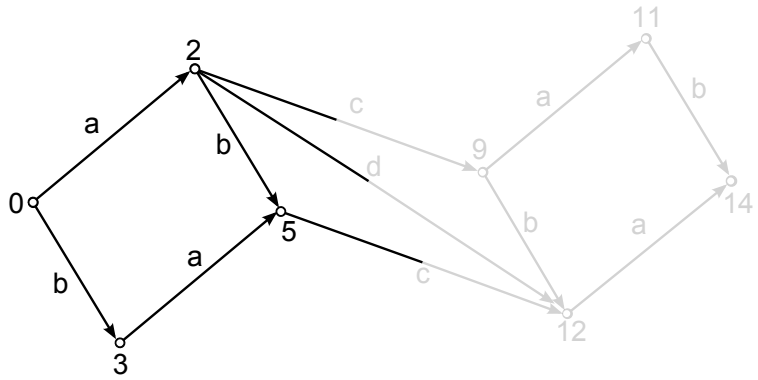


Figure 2.5: Prefix path $x_0 = 0, x_1 = 2$ and suffix path $x_0 = 0, x_2 = 3, x_3 = 5$ for Example 2.1. These paths form a valid pair.

implies that we cannot hope to find an efficient algorithm for the Antisymmetric Longest Path problem in general, unless P = NP. Here, "efficient" means an algorithm with polynomial running time. But the particular structure of spectrum graphs allows us to find such an efficient algorithm, which will be presented in the next section.

One can easily come up with a naïve algorithm to recover the peptide string from the set of masses $\mathcal{M}$: For every pair x_j, y_j we decide whether x_j or y_j is part of the path through $G(\mathcal{M})$. Obviously, there are 2^n possibilities for this. We then compute the graph induced by the selected nodes, and we test whether the induced graph contains a path from 0 to M through all nodes, which can be done in linear time. If so, then the resulting string s satisfies $\mathcal{M}(s) = \mathcal{M}$ and we are done. The running time of this algorithm is $O(2^n \cdot n)$ and, hence, the algorithm is limited to rather small strings. In practice, running times of this naïve algorithm are probably acceptable for up to 20 peaks, but are prohibitive in applications as soon as we drop our simplifying assumptions. In practice, we can speed up the algorithm by building the string s from left to right, deciding on which peaks belong to the prefix path as we go. Using this branch-and-bound search, we can discard prefixes that cannot result in an admissible string, see Exercise 2.5. Early algorithms for the peptide *de novo* sequencing problem were in fact based on the branch-and-bound search paradigm. Unfortunately, there is no simple way to generalize this approach for additional and missing peaks, see Sec. 2.5.4.

2.4 Dynamic programming for ideal data

We are given a set of masses $\mathcal{M}$ with spectrum graph $G := G(\mathcal{M})$; our task is to find a valid path in $G = (V, E)$. From the above, we may assume that $V = \mathcal{M} = \{x_0, \ldots, x_n, y_n, \ldots, y_0\}$ satisfying (2.2).

We could now directly search for a valid path p in G. This has the conceptual disadvantage that we always have to consider pairs x_i, y_i where exactly one of the two nodes is part of the path. So, we use a detour that is conceptually slightly simpler: We ignore $y_n, \ldots, y_0$ and concentrate on nodes $x_0, \ldots, x_n$. We construct *two* paths in the spectrum graph called *prefix path* and *suffix path*, both starting in $x_0 = 0$. We require that these two paths are node-disjoint, with the exception of the start node $x_0 = 0$. Furthermore, each node $x_1, \ldots, x_n$ must be part of either the prefix path or the suffix path. Consequently, we say that a prefix path to x_i and a suffix path to x_j form a *valid pair* if, for all $l = 1, \ldots, \max\{i, j\}$, node x_l is either an element of the suffix path or of the prefix path. See Fig. 2.5 for two paths that form a valid pair for Example 2.1.

What is the connection between a valid path, and a valid pair of prefix and suffix path? Let p be a valid path in G. Let p_1 be the part of p with nodes from $x_0, \ldots, x_n$, and let p_2 be the remaining part of the path with nodes from $y_n, \ldots, y_0$. Now, we can *flip* $p_2 = v_0 \ldots v_l$ by setting $p_2^* := u_l \ldots u_0$ with $u_j := M - v_j$ for all $j = 1, \ldots, l$. One can easily check that p_2^* is a path in G, and that p_1 and p_2^* form a valid pair of paths.

On the other hand, assume that we are given a valid pair of a prefix path p_1 to x_i and suffix path p_2 to x_j. Analogously to above, we flip p_2 to generate a path p_2^* that uses only nodes from $y_n, \dots, y_0$. In order to "glue" together these two paths, we have to make sure that they can be connected via an edge: We know that x_i is the last node of the prefix path, and that y_j is the first node of the flipped suffix path. If $x_i y_j$ is an edge of the spectrum graph, then we can connect the two paths p_1 and p_2^*, resulting in a single path p. Is this path valid? Not necessarily so: Clearly, either x_l or y_l is a node of p, for all $l = 1, \dots, \max\{i, j\}$. But we also have to make sure that all nodes of the spectrum graph are part of the path: This is the case if and only if $\max\{i, j\} = n$ holds.

We want to use dynamic programming to test whether our instance has a solution, see Sec. 15.5. We define a binary matrix $D[0 \dots n, 0 \dots n]$ as follows: We set $D[i, j] = 1$ if there is a prefix path from x_0 to x_i and a suffix path from x_0 to x_j that form a valid pair; and $D[i, j] = 0$ otherwise.[3] Clearly, $D[0, 0] = 1$ holds, as well as $D[j, j] = 0$ for $j = 1, \dots, n$. We will use this to initialize the *main diagonal* $D[j, j]$ of our matrix. Also note that the matrix D is symmetric, so $D[i, j] = D[j, i]$ holds for all i, j.

Example 2.2. Consider the weighted alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ and the fragmentation spectrum $\mathcal{M} := \mathcal{M}(s) = \{0, 2, 3, 5, 9, 11, 12, 14\}$ from Example 2.1. See Fig. 2.5 for the spectrum graph. Then, the matrix D is:

	$j=0$	1	2	3
$i=0$	1	1	0	0
1	1	0	1	1
2	0	1	0	1
3	0	1	1	0

For example, $D[2, 3] = 1$ tells us that there exists a prefix path to x_2 and a suffix path to x_3 that form a valid pair; namely, this prefix path is $x_0 x_2$, and the suffix path is $x_0 x_1 x_3$. Exchanging prefix path and suffix path, we also have $D[3, 2] = 1$.

But how can we efficiently compute matrix D? Consider any entry $D[i, j]$: If $i \geq j + 2$ then $i - 1 > j$, so the node x_{i-1} cannot be part of the suffix path ending in x_j. Hence, $D[i, j] = 1$ holds if and only if $D[i - 1, j] = 1$ and $x_{i-1} x_i$ is an edge of the spectrum graph. Analogously, for $i \leq j - 2$ we have $D[i, j] = 1$ if and only if $D[i, j - 1] = 1$ and $x_{j-1} x_j \in E$.

So, the only entries $D[i, j]$ of the matrix we are left with to compute, are those with $i = j+1$ or $i = j-1$. The corresponding elements $D[i, i - 1]$ and $D[i, i + 1]$ are called *secondary diagonals*. We concentrate on the first case $i = j + 1$. Here, the prefix path ends in x_i and the suffix path ends in $x_j = x_{i-1}$, so the previous nodes x_l for $l = 1, \dots, i - 2$ can be part of either the prefix or the suffix path. Assume that $D[i, j] = 1$, so there is a valid pair of prefix and suffix path. We consider all possible last edges of the prefix path: Obviously, the prefix path must end with *some* edge $x_l x_i \in E$ for $l \in \{0, \dots, i - 2\}$. In addition, there must be a prefix path to x_l and a suffix path to x_j that form a valid pair. But the latter is true, by definition of D, if and only if $D[l, j] = 1$. So, for all $l = 0, \dots, i - 2$, we test if $D[l, j] = 1$ and $x_l x_i \in E$ holds simultaneously; if we find one such l, then $D[i, j] = 1$. The case $i = j - 1$ can be solved analogously.

The above argument actually proves that the following recurrence is correct:

$$D[i,j] = \begin{cases} D[i-1,j] & \text{if } i \geq j+2 \text{ and } x_{i-1}x_i \in E \\ D[i,j-1] & \text{if } j \geq i+2 \text{ and } x_{j-1}x_j \in E \\ \max_{l=0,\dots,j-1}\{D[l,j] : x_l x_{j+1} \in E\} & \text{if } i = j+1 \\ \max_{l=0,\dots,i-1}\{D[i,l] : x_l x_{i+1} \in E\} & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

We initialize $D[0, 0] = 1$ and $D[j, j] = 0$ for all $j \geq 1$, and we set $D[1, 0] = D[0, 1] = 1$ if $x_0 x_1 \in E$, and $D[1, 0] = D[0, 1] = 0$ otherwise; recurrence (2.3) is used for all remaining entries. In (2.3) we assume that

[3] Note that this (and only this) is the *definition* of the matrix D, whereas Eq. (2.3) below is a *recurrence* that tells us how to compute D.

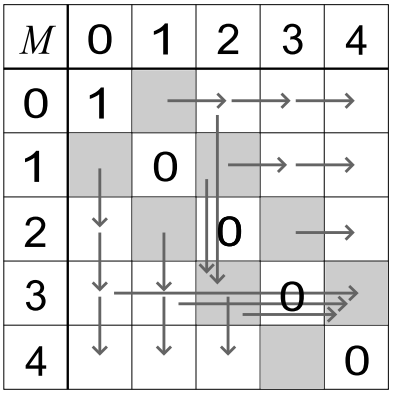


Figure 2.6: Illustration of how recurrence (2.3) accesses entries in the matrix.

$\max \emptyset = 0$, to simplify the formalism. Note that taking the maximum is only some "math voodoo", which allows us to write up the equation more easily: The expression evaluates to one if at least one of the entries in the set is non-zero. See Fig. 2.6 on how the recurrence accesses other entries in the matrix.

How much time do we need to compute D? The main diagonal is initialized in $O(n)$ total time. For every entry of the two secondary diagonals $i = j + 1$ and $i = j - 1$, computing the maximum requires $O(n)$ time. As there are $O(n)$ entries on the secondary diagonals, this leads to $O(n^2)$ time in total. All other entries can be computed in constant time; as there are $O(n^2)$ entries remaining, this again results in $O(n^2)$ time. In total, we need $O(n^2)$ time to compute the complete matrix D. Obviously, we need $O(n^2)$ memory to store the matrix D; this requirement can be reduced to $O(n)$, see Exercise 2.6.

The actual computation of the matrix is quite simple: To compute $D[i, j]$, recurrence (2.3) accesses only entries $(i', j') \neq (i, j)$ such that $i' \leq i$ and $j' \leq j$ holds. So, we can fill the matrix from the upper-left entry to the lower-right entry. We show the resulting "algorithm" in Algorithm 2.1.

Having computed the matrix D, how does that help us to check if there is a valid path? Assume that $D[i, j] = 1$ holds for some i, j with $\max\{i, j\} = n$. By definition of D, this means that there is a prefix path to x_i and a suffix path to x_j that form a valid pair. We can flip the suffix path, as described above; to glue together the two resulting paths, we only have to check if $x_i y_j$ is an edge. The resulting path is valid, since $\max\{i, j\} = n$ holds. Consequently, we check if there is some $i \in \{1, \ldots, n\}$ with $D[i, n] = 1$ and $x_i y_n \in E$, or some $j \in \{1, \ldots, n\}$ with $D[n, j] = 1$ and $x_n y_j \in E$. If we can find such i or j, then there is a valid path in the spectrum graph and, consequently, also a string $s \in \Sigma^*$ with $\mathcal{M}(s) = \mathcal{M}$; but if there is no such i or j, then there is also no valid path and, hence, no such string. I have integrated this query into Algorithm 2.1.

So, our DP matrix lets us decide if there is at least one string s such that $\mathcal{M}(s) = \mathcal{M}$; but, how do we recover this string? The answer to this question is *backtracing*. In principle, we could proceed in three steps: First, we use the matrix D to recover a valid pair of prefix path and suffix path, which can be glued into a valid path in the spectrum graph. Then, we can flip the suffix path to construct the valid path. Finally, we transform the valid path into a string.

But it is possible to do all three steps at once, directly constructing the string s. Assume that $D[i, j] = 1$ holds for indices i, j with $\max\{i, j\} = n$, and that $x_i y_j \in E$. We simultaneously extend the string s to the left and to the right. We initialize $s \leftarrow z$ for the unique character $z \in \Sigma$ with $\mu(z) = y_j - x_i$. Looking at recurrence (2.3), we see that "$D[i, j] = 1$" has progressed from some entry $D[i', j']$ with either $i' < i$ and $j' = j$, or $i' = i$ and $j' < j$. In case $i' < i$ and $j' = j$, we append the unique character $z \in \Sigma$ with $\mu(z) = x_i - x_{i'}$ to the left side of s. In the other case $i' = i$ and $j' < j$, we append the unique character $z \in \Sigma$ with $\mu(z) = y_{j'} - y_j = x_j - x_{j'}$ to the right side of s. Let $(i, j) \leftarrow (i', j')$ and repeat, until we reach the upper-left entry $(i, j) = (0, 0)$. Now, s is the string we are searching for, satisfying $\mathcal{M}(s) = \mathcal{M}$: In fact, we have constructed a valid path by our backtracing procedure, so this follows directly from Lemma 2.1. See Algorithm 2.2 for the pseudocode of this algorithm.

function PEPTIDESEQUENCINGIDEALDATA(set of masses $\mathcal{M}$)
  Test that $\mathcal{M}$ has even cardinality
  Let $\{x_0, \ldots, x_n, y_n, \ldots, y_0\} := \mathcal{M}$ satisfying (2.2)
  Let $M := y_0$
  Test $x_i + y_i = M$ for all $i = 0, \ldots, n$
  Construct spectrum graph $G = (V, E)$ from $\mathcal{M} = \{x_0, \ldots, x_n, y_n, \ldots, y_0\}$
  Init binary matrix $D[0 \ldots n, 0 \ldots n]$ with $D[0,0] \leftarrow 1$, $D[i,i] \leftarrow 0$ for $i = 1, \ldots, n$, and $D[1,0], D[0,1] \leftarrow 1$ if $x_0 x_1 \in E$ and 0 otherwise
  for $i \leftarrow 0, \ldots, n$ **do** ▷ Fill the matrix
    for $j \leftarrow 0, \ldots, n$ **do**
      if $i \neq j$ and $\max\{i, j\} \geq 2$ **then**
        Compute $D[i, j]$ from (2.3)
      end if
    end for
  end for
  for $i \leftarrow 0, \ldots, n$ **do** ▷ Check if there is a valid path
    if $D[i, n] = 1$ and $x_i y_n \in E$ **then**
      return (i, n)
    end if
  end for
  for $j \leftarrow 0, \ldots, n$ **do**
    if $D[n, j] = 1$ and $x_n y_j \in E$ **then**
      return (n, j)
    end if
  end for
  return *false*
end function

Algorithm 2.1: Peptide *de novo* sequencing for ideal data: We first compute the matrix D using recurrence (2.3); then check if there is a valid path using this matrix.

2.5 Getting rid of the unrealistic assumptions

Hitherto, we have used some unrealistic assumptions to simplify the problem, which we will abandon in the following. We will see that we have succeeded in "making the problem as simple as possible, but not simpler": We have to replace recurrence (2.3) by an optimization version which is even a bit simpler. But besides playing around with some weights, no other changes are required.

Our presentation will touch upon certain issues that will be covered in more detail at a later stage of this textbook. After all, this is only the beginning of our journey through the realms of computational mass spectrometry. These issues include:

- The general problem of matching mass spectra; this will be covered in Sec. 4.5.
- Penalizing (or rather not penalizing) additional peaks in Sec. 2.5.1; see also Sec. 4.6.
- Penalizing missing peaks in Sec. 2.5.3; we will come back to this in Sec. 4.5.
- "Strings without order" in Sec. 2.5.3; this leads to the definition of "compomers" in the next chapter.

2.5.1 Additional peaks

The next problem that we want to deal with, is that the set of masses $\mathcal{M}$ contains additional peaks: In practice, we will not only record the masses of prefixes and suffixes of our peptide string but, in addition,

function BACKTRACINGIDEALDATA(matrix $D[0 \dots n, 0 \dots n]$, integers i, j, graph $G = (V, E)$)
  Let $\{x_0, \dots, x_n, y_n, \dots, y_0\} := V$ satisfying (2.2)
  Assure that $\max\{i, j\} = n$, $D[i, j] = 1$, and $x_i y_j \in E$
  Choose $z \in \Sigma$ with $\mu(z) = y_j - x_i$
  Let $s \leftarrow z$
  while $(i, j) \neq (0, 0)$ **do** ▷ Backtracing starts here
    Find (i', j') such that $D[i, j] = D[i', j']$ in (2.3)
    if $i' < i$ **then** ▷ implies $j' = j$
      Choose $z \in \Sigma$ with $\mu(z) = x_i - x_{i'}$
      $s \leftarrow zs$ ▷ extend prefix part of the string
    else ▷ implies $i' = i$ and $j' < j$
      Choose $z \in \Sigma$ with $\mu(z) = y_{j'} - y_j$
      $s \leftarrow sz$ ▷ extend suffix part of the string
    end if
    Let $i \leftarrow i'$ and $j \leftarrow j'$
  end while
  Return s
end function

Algorithm 2.2: Peptide *de novo* sequencing for ideal data: We backtrace through the matrix D to reconstruct the string s. We assume that the spectrum graph G as well as indices i, j to start the backtracing, have been computed beforehand.

many peak masses that do not correspond to our peptide at all. Furthermore, peptide fragmentation is more complex than what we have described above, and peak masses in $\mathcal{M}$ may stem from fragments that we have not accounted for in our simple model. Be aware that in this section, we assume no peaks to be missing; so, $m \in \mathcal{M}$ still implies $M - m \in \mathcal{M}$. This means that we can still name our peak masses $\mathcal{M} = \{x_0, \dots, x_n, y_n, \dots, y_0\}$ satisfying (2.2).

How can we decide which string is the "best" one? An obvious choice is the following: We say that a string s *explains* some peak mass m if $m \in \mathcal{M}(s)$. Now, the string that explains the maximum number of peaks in the measured set of peaks $\mathcal{M}$, is a natural choice for this "best" answer.

At this point, two things must be understood. If our set of masses does not contain any additional peaks, then a string explaining a maximum number of peaks, is also a solution of the ideal problem without additional peaks, and vice versa: This string explains *all* the peaks in the spectrum, which is obviously optimal. This is not only a nice gimmick, but rather a necessity: If you transform an algorithm for idealized data into its optimization version, then the optimization-based algorithm should come up with the correct solution if you feed it with ideal data. This is true here, so we can move on. Here comes the second important point: If the set of masses contains additional peaks and if we are unlucky, then the string that explains a maximum number of peaks is not the true string that the fragmentation spectrum stems from. But this is not a particular problem of our approach, but rather a general one for any method that has to deal with noisy data: In case the quality of the data is bad, no computational method in the world will be able to reconstruct the true string. Several times throughout this textbook, we will find that there is no way around "rubbish in — rubbish out". All that we can do, is try to push the limits of what we consider "rubbish" as far as possible.

We define a matrix $Q[0 \dots n, 0 \dots n]$ where $Q[i, j]$ is the maximum number of peaks explained by the prefix path x_0 to x_i and the suffix path x_0 to x_j, such that these paths form a valid pair. We will ignore the peaks at masses 0 and M, as these are not informative. Note that we are not penalizing for the presence of additional peaks; this will be discussed (and justified) in Sec. 4.6. We initialize $Q[0, 0] = 0$ and $Q[j, j] = -\infty$ for $j = 1, \dots, n$. Here, $Q[i, j] = -\infty$ means that the solution is invalid, so there is no valid pair of paths to x_i and x_j.

How can we compute the maximum number of peaks explained by any string, if we know the matrix Q? Unlike for ideal data, the optimal valid path may skip the node pair x_n, y_n altogether. Therefore, we iterate over all edges $x_i y_j \in E$, and search for the maximum value $Q[i,j]$. Then, $2Q[i,j]$ is in fact the maximum number of peaks that can be explained by any string, ignoring masses 0 and M, see Exercise 2.9. We have to multiply by two, as a peak m explained by the prefix path will also explain the corresponding peak $M - m$, and similarly for the suffix path.

To simplify the recurrence for Q, we define a *scoring function* w which, for the moment, will only be used to count peaks. At a later stage, we will reuse the scoring function to encode more complex things. In addition, we use w to encode whether or not an edge uv is present in the spectrum graph $G = (V, E)$. To this end, we define

$$w(x,y) := \begin{cases} 1 & \text{if } xy \in E, \\ -\infty & \text{otherwise.} \end{cases} \tag{2.4}$$

Now, $w(u,v) = 1$ holds if and only if $uv \in E$. We can think of w as (unit) edge weights to the spectrum graph.

Introducing $-\infty$ as a score, allows us to come up with a very simple recurrence for Q, similar to (2.3). For readers not familiar with calculations involving $\pm\infty$, we note that $x + -\infty = -\infty$ and $x > -\infty$ holds for all numbers $x \in \mathbb{R}$. The recurrence for Q is:

$$Q[i,j] = \begin{cases} \max_{l=0,\dots,i-1}\{Q[l,j] + w(x_l, x_i)\} & \text{if } i > j \\ \max_{l=0,\dots,j-1}\{Q[i,l] + w(y_j, y_l)\} & \text{if } j > i \end{cases} \tag{2.5}$$

At this point, our scoring function w serves a single purpose: Entries $Q[l,j]$ and $Q[i,l]$ are not taken into consideration for the maxima in (2.5) if $x_l x_i \notin E$ or $y_j y_l \notin E$ holds, respectively. Note that we have deliberately broken the symmetry, accessing $w(y_j, y_l)$ instead of $w(x_l, x_j)$ in the recurrence. At present, we have $w(y_j, y_l) = w(x_l, x_j)$; but we will see in the next section that it can be reasonable to define a non-symmetric scoring function in practice.

We now show that recurrence (2.5) is correct. Consider entry $Q[i,j]$; we concentrate on the case $i > j$, the other case follows analogously. Let $E' \subseteq E$ be the set of edges ending in x_i. Clearly, $w(x'x_i) = 1$ holds for all $x'x_i \in E'$. If E' is empty then there is no prefix path ending in x_i, so $Q[i,j] = -\infty$ is correctly calculated by (2.5). If $Q[l,j] = -\infty$ holds for all entries in the maximum from (2.5), then there is no valid pair of paths to x_j and any predecessor of x_i; again, $Q[i,j] = -\infty$ is correctly calculated. In the following, we assume $Q[i,j] \neq -\infty$; this implies that there is a prefix path to x_i and a suffix path to x_j forming a valid pair.

Assume that $x_L x_i$ is the last edge of the optimal prefix path. By induction, $Q[i,j] = Q[L,j] + 1$ must hold, as our new prefix path explains exactly one more peak. This implies $Q[i,j] \leq \max_{l=0,\dots,i-1}\{Q[l,j] + w(x_l, x_i)\}$, and it remains to be shown that $Q[L,j] = \max_{l=0,\dots,i-1}\{Q[l,j] : x_l x_i \in E\}$. This follows as otherwise, $x_L x_i$ would not be the last edge of an optimal prefix path, in contradiction to our assumption. This concludes our proof. □

Compare (2.3) with (2.5): The second recurrence appears to be simpler than the first one. This is because we no longer treat the two secondary diagonals differently; instead, we have to compute maxima for all elements of the matrix, as extending either prefix or suffix path is always possible, assuming all unexplained peaks to be additional. But as so often, our intuition is misleading: Whereas the second recurrence appears simpler, its computation takes more time. In fact, it is quite easy to see that computing the complete matrix Q requires $O(n^3)$ time, and we need $O(n^2)$ memory to store it. So, running time has increased from quadratic for ideal data, to cubic when additional peaks have to be taken into account. Also, there is no way to reduce memory requirements to $O(n)$, see Exercise 2.6. From the theoretical standpoint, this is a huge increase in running time; luckily, n is rather small in practice with $n \leq 100$ in most cases, so computation time will hardly ever reach one second on a modern computer.

Again, we are left with the task of recovering the optimal solution from the matrix Q. Similar to above, this is achieved by backtracing, this time through the matrix Q: Assume that $Q[i,j]$ with $x_i y_j \in E$ is

maximum. We start with $s = x$ for $x \in \Sigma$ with $\mu(x) = y_j - x_i$. We search for $Q[i', j']$ where $Q[i, j]$ in the maxima of (2.5) has progressed from, so $Q[i, j] = Q[i', j'] + w(x_{i'}, x_i)$ or $Q[i, j] = Q[i', j'] + w(y_j, y_{j'})$. We again have two cases, appending a character either to the left end or the right end of s. We set $(i, j) \leftarrow (i', j')$ and repeat, until we reach $(i, j) = (0, 0)$.

2.5.2 General edge-weighted spectrum graphs

In the previous section, the edge weighting w was merely a trick, so that we did not have to treat edges and "non-edges" of G separately in recurrence (2.5). But it turns out that exactly the same recurrence can be used in case the spectrum graph is edge-weighted: Given a graph $G = (V, E)$, any function $w : E \to \mathbb{R}$ is an *edge weighting*. The *weight* (or *length*) of a path $p = u_0 u_1 \dots u_l$ in G is then simply the sum of edge weights,

$$w(p) := \sum_{i=1}^{l} w(u_{i-1} u_i).$$

Now, the following lemma tells us that we can use recurrence (2.5) to search for longest valid paths:

Lemma 2.2. *Let $G = (V, E)$ be a spectrum graph for some set of masses $\mathcal{M} = \{x_0, \dots, x_n, y_n, \dots, y_0\}$ with $x_i + y_i = M$ for all $i = 0, \dots, n$. Let $w : E \to \mathbb{R}$ be arbitrary edge weights, and set $w(x, y) := -\infty$ for $xy \notin E$. Then, the maximum weight of a valid path in G from 0 to M, equals $\max\{Q[i, j] + w(x_i, y_j) : x_i y_j \in E\}$ where Q is computed using (2.5).*

We leave the proof of the lemma to the reader, see Exercise 2.10. For this proof, we have to formally define the dynamic programming matrix Q. This is slightly more complicated than above: Given a prefix path p_1 and a suffix path p_2 in G, we say that the *length* of this pair is $w(p_1) + w(p_2^*)$. The important point to note is that we are not using the weight of the suffix path p_2 itself but instead, we use its flipped counterpart. Now, we can formally define $Q[i, j]$ to be the maximum length of a prefix path to x_i and a suffix path to x_j that form a valid pair.

Looking at the lemma, you will notice one important change: The maximum weight of a valid path is $\max\{Q[i, j] + w(x_i, y_j) : x_i y_j \in E\}$ whereas previously, we searched for $\max\{Q[i, j] : x_i y_j \in E\}$. Where does the additional weight of $w(x_i, y_j)$ come from?

There are two answers to this question. The first answer is formal and simple: From the definition of Q we see that the weight of the edge $x_i y_j$ connecting the prefix path p_1 with the flipped suffix path p_2^* has not been added yet. So, in our maximization, we simply take care of that missing edge.

The second answer is somewhat harder to explain: The weight of a path is defined as the sum of edge weights. But what we want to score in our mass spectrum, are peaks; and peaks correspond to nodes, not edges! The conceptually most elegant way to get around this dilemma, is to push the weight of a node to all of its incoming edges. In this way, weight $w(u, v)$ corresponds to the weight of node v. As at most one of the edges entering v is part of the path, we add the weight of a node if and only if the path passes through that node.

This is different from our initial definition of the matrix Q, because now, a suffix path from 0 to x_j no longer explains the peak y_j: The first edge of the flipped suffix path is $y_j y$ for some node $y > y_j$, and $w(y_j, y)$ tells us something about the node (and peak) y, but *not* about y_j. Combining the peak counting score (2.4) with Lemma 2.2 leads to the following interpretation: Entry $Q[i, j]$ is the maximum number of peaks from $\mathcal{M} \setminus \{0\}$ explained by a prefix path to x_i and a suffix path to x_j that form a valid pair, ignoring x_j in our calculation. And we reach $\max\{Q[i, j] + w(x_i, y_j) : x_i y_j \in E\}$ as the maximum number of peaks from $\mathcal{M} \setminus \{0\}$ that can be explained by a valid path. (Note that we deliberately excluded 0 but not M from being counted.) As this definition is much harder to grasp than what we initially came up with, the reader will hopefully excuse our little detour.

2.5.3 Missing peaks

Peaks can be missing because the corresponding fragment was not formed, or not detected; but also because the peak picking software decided not to report it, see Sec. 13.1. First, assume that either some prefix

peak m or the complementing suffix peak $M-m$ is missing, but never both at the same time. In this case, we can "reconstruct" the missing information by mirroring the spectrum, $\mathcal{M}' := \mathcal{M} \cup \{M-m \,:\, m \in \mathcal{M}\}$. We assume $\mathcal{M}' = \{x_0, \ldots, x_n, y_n, \ldots, y_0\}$ satisfying (2.2). We construct our spectrum graph using the set $\mathcal{M}'$ instead of $\mathcal{M}$. For counting peaks, we define the score w by:

$$w(x,y) := \begin{cases} 0 & \text{if } xy \in E,\ y \notin \mathcal{M}, \text{ and } M-y \notin \mathcal{M} \\ 1 & \text{if } xy \in E, \text{ and either } y \in \mathcal{M} \text{ or } M-y \in \mathcal{M} \\ 2 & \text{if } xy \in E,\ y \in \mathcal{M}, \text{ and } M-y \in \mathcal{M} \\ -\infty & \text{if } xy \notin E \end{cases} \tag{2.6}$$

The nice thing is that recurrence (2.5) can be applied without changes, see Lemma 2.2. Here, $\max\{Q[i,j] + w(x_i, y_j) \,:\, x_i y_j \in E\}$ is the maximum number of peaks that can be explained by any string, including masses 0 and M.

Up to this point, both the scoring function w as well as the resulting matrix Q have been symmetric. But the fact that peaks may be missing, is a reason to break this symmetry: It is possible that in practice, the presence of a prefix peak (b ion) is seen as more informative than the presence of a suffix peak (y ion). So, seeing the prefix peak but not the suffix peak, is "better" than seeing the suffix peak but not the prefix peak. Instead of simply counting explained peaks, we may want to define a different score: We take twice the number of peaks explained by prefixes, plus the number of peaks explained by suffixes. Then, we can define a scoring function w' as:

$$w'(x,y) := \begin{cases} 0 & \text{if } xy \in E,\ y \notin \mathcal{M}, \text{ and } M-y \notin \mathcal{M} \\ 1 & \text{if } xy \in E,\ y \notin \mathcal{M}, \text{ but } M-y \in \mathcal{M} \\ 2 & \text{if } xy \in E,\ y \in \mathcal{M}, \text{ but } M-y \notin \mathcal{M} \\ 3 & \text{if } xy \in E,\ y \in \mathcal{M}, \text{ and } M-y \in \mathcal{M} \\ -\infty & \text{if } xy \notin E \end{cases}$$

Again, recurrence (2.5) can still be applied without changes, see Lemma 2.2, and Exercise 2.12 for an even more general approach.

The case where prefix and suffix peak at m and $M-m$ are simultaneously missing, is only slightly more complicated: We can simply check if the mass difference between two peaks can be explained as the mass of up to k amino acid residues, where $k \in \mathbb{N} \cup \{\infty\}$ is a fixed parameter set by the user. We can think of this as inserting additional edges into the spectrum graph $G = (V, E)$: For $u, v \in V$ there is an edge $uv \in E$ if and only if there exists some $z \in \Sigma^*$ with $1 \le |z| \le k$ such that $u + \mu(z) = v$. A theoretically more elegant way, is to replace our original weighted alphabet Σ by an extended version Σ', which contains a character for every non-empty string of the original alphabet Σ with up to k characters. We then have to delete characters from Σ' that have identical mass. We will not pursue this "elegant way"; but it implies that all of our definitions and results for spectrum graphs, are still valid if we insert the additional edges.

Note that we cannot infer the order of characters inside the "gap string" z; we will come up with a formalism for this situation in the next chapter, where we introduce compomers as "strings without order". In the literature, this situation is often denoted as $s = \mathsf{a[bc]d}$, meaning that we have no information whether the true string is abcd or acbd. If the gap gets so large that the mass can be explained by more than one combination of characters, we can use the notation $s = \mathsf{a[186]d}$ for a gap of 186 Da.

To our delight, matrix Q and recurrence (2.5) from above can be used without any changes. This follows using our idea of an "extended alphabet" Σ' and Lemma 2.2. Again, we are searching for the string that explains a maximum number of peaks. This should be combined with our approach of mirroring the spectrum, to reconstruct missing prefix and suffix peaks.

Larger steps in the spectrum graph explain fewer peaks, so we force the approach to use these larger steps as prudently as possible. Unfortunately, that is not quite the end of the story. Let $\Sigma = \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\}$ be

the weighted alphabet from Example 2.1. Assume that $s = \mathsf{cd}$ is the correct peptide string; for ideal data, we have $\mathcal{M} = \{0, 7, 10, 17\}$. Obviously, the string s explains all of these masses; but so does the string $s' = \mathsf{caaaaa}$ with

$$\mathcal{M}(s') = \{0, 2, 4, 6, 7, 8, 9, 10, 11, 13, 15, 17\}.$$

It is understood that we should be able to distinguish between s and s' based on this data. We can do so by penalizing unobserved (missing) peak pairs, where both the prefix and the suffix peak are missing from the measured mass spectrum. Again, we do not have to change the recurrence, but simply modify the scoring w: For example, we may modify the score for counting peaks from (2.6) by defining

$$w(x, y) := -\min\{|z| - 1 \,:\, z \in \Sigma^*, \mu(z) = y - x\} \tag{2.7}$$

for the case $xy \in E$, but $y \notin \mathcal{M}$ and $M - y \notin \mathcal{M}$. Then, a "gap string" $z = z_1 z_2$, where we cannot find a prefix or suffix pair for appending either z_1 or z_2, is penalized by -1 for one missing peak pair. If there are multiple gap strings that can bridge the gap, then we have to give the string the benefit of the doubt, penalizing it the least. The nice thing is that recurrence (2.5) can still be applied without changes. How do we find the minimum length of a string that explains some mass difference? This will be addressed in the next chapter, see Exercise 3.1.

We leave the proof that all of these calculations and recurrences are in fact correct, to the reader, see Exercise 2.14. In Sec. 4.5, we will come back to the problem of penalizing missing peaks.

2.5.4 Prefix mass equals suffix mass

Before we discuss how to get rid of Assumption 3, we want to take a short detour and explain why this assumption was introduced in the first place. Assume that we want to maximize the number of explained peaks, ignoring missing peaks. Initially, people did not consider the number of explained peaks to find the "best" solution, as this is somewhat complicated to compute. Instead, they looked at a simpler score that, for some string $s \in \Sigma^*$, counts the number of proper prefix masses of s present in $\mathcal{M}$, plus the number of proper suffix masses of s present in $\mathcal{M}$. This score is easy to incorporate into branch-and-bound approaches, as it allows us to truncate the search space.

Consider the "true" string $s = \mathsf{aaabb}$ for the weighted alphabet from Example 2.1. Let $\mathcal{M} := \mathcal{M}(s) = \{0, 2, 3, 4, 6, 8, 9, 10, 12\}$ be the ideal fragmentation spectrum of s. Now, the true string s has four proper prefixes and four proper suffixes, all of which are present in $\mathcal{M}$, leading to a score of 8. Where is the problem? Consider the string $s' = \mathsf{aaaaaa}$: This string has five proper prefixes and five proper suffixes, all of which are present in $\mathcal{M}$, resulting in score 10. So, we have found a string that better explains the data than the true solution! Obviously, this is not the case, the problem being *peak double counting*: We have counted each peak $2, 4, 6, 8, 10$ twice in our scoring, although these peaks are present only once in $\mathcal{M}$. In fact, the string s' explains only seven out of nine peaks in $\mathcal{M}$. So, we should be able to tell that this explanation is worse, without having to rely on scoring missing peaks. Note that even the true string s counts peak 6 twice, as prefix aaa and as suffix bb; see Exercise 2.13 for an example where the true string does not double count.

Demanding that any string s must not contain a proper prefix and suffix of identical mass, altogether removes the problem: No longer can a peak be scored twice, as all proper prefixes and suffixes are required to have different masses. But this comes at the price of a reduced generalizability of the method: Certain strings can simply not be found, even if they are the correct answer. For our simplified model, it is quite obvious that many strings violate Assumption 3: Any string that contains a prefix a and a suffix b with the same composition of characters, violates our assumption. But it is also true in practice, see Exercises 3.13 and 3.14. We will now show how to get around peak double counting without artificially limiting the search space.

So, let us drop Assumption 3. To simplify our presentation, we limit our considerations to the case of "additional peaks only", resulting in the simplest scoring function $w_{\mathrm{A}} \equiv w$ with $w(x, y) \in \{1, -\infty\}$ from (2.4). We will deliberately *not* use a general scoring function in our presentation; we will come back

function PEPTIDESEQUENCING(set of masses $\mathcal{M}$, precursor mass M)
  Let $\mathcal{M}' := \{0, M\} \cup \{m, M - m \,:\, m \in \mathcal{M}\}$
  Let $\{x_0, \ldots, x_n, y_n, \ldots, y_0\} := \mathcal{M}'$ satisfying (2.2)
  Construct spectrum graph $G = (V, E)$ from $\mathcal{M}'$
  Matrix $Q'[0 \ldots n, 0 \ldots n]$
  Init $Q'[0, 0] \leftarrow 0$
  for $i \leftarrow 0, \ldots, n$ **do** ▷ Fill the matrix
    for $j \leftarrow 0, \ldots, n$ **do**
      if $(i, j) \neq (0, 0)$ **then**
        Compute $Q'[i, j]$ from (2.8)
      end if
    end for
  end for
  Let *max score* $\leftarrow -\infty$ ▷ Check if there is a valid path
  for $i \leftarrow 0, \ldots, n$ **do**
    for $j \leftarrow 0, \ldots, n$ **do**
      if $x_i y_j \in E$ and $Q'[i, j] >$ *max score* **then**
        Let *max score* $\leftarrow Q'[i, j]$ and $(i', j') \leftarrow (i, j)$
      end if
    end for
  end for
  Return (i', j') with score *max score*
end function

Algorithm 2.3: Peptide *de novo* sequencing with additional peaks: We first compute the matrix Q' using recurrence (2.8); then search for the path through the spectrum graph with highest score.

to this issue later. We define a matrix $Q'[0 \ldots n, 0 \ldots n]$ where $Q'[i, j]$ is the maximum number of peaks in $\mathcal{M} \setminus \{0, M\}$ explained by any prefix path x_0 to x_i and suffix path x_0 to x_j. Here, nodes that are present in both the prefix path and the suffix path are counted only once; but we no longer ask that prefix path and suffix path form a valid pair. The initialization is reduced to $Q'[0, 0] = 0$, since $Q[j, j] = -\infty$ only made sense when Assumption 3 was still in place.

We reuse recurrence (2.5) for Q', but extend it by the calculation of the diagonal matrix elements:

$$Q'[i, j] = \begin{cases} \max_{l=0,\ldots,i-1} \{Q'[l, j] + w_{\mathrm{A}}(x_l, x_i)\} & \text{if } i > j \\ \max_{l=0,\ldots,j-1} \{Q'[i, l] + w_{\mathrm{A}}(y_j, y_l)\} & \text{if } j > i \\ \max_{l=0,\ldots,i-1} \{Q'[l, j] \,:\, x_l x_i \in E\} & \text{if } i = j \end{cases} \tag{2.8}$$

Once more, recall that $w_{\mathrm{A}}(x, y) = 1$ for $xy \in E$, and $w_{\mathrm{A}}(x, y) = -\infty$ otherwise. Also recall that we assume $\max \emptyset = -\infty$. The last case of the recurrence appears to be non-symmetric; but this is due to the fact that

$$\max_{l=0,\ldots,i-1} \{Q'[l, j] \,:\, x_l x_i \in E\} = \max_{l=0,\ldots,j-1} \{Q'[i, l] \,:\, y_j y_l \in E\}. \tag{2.9}$$

See Exercise 2.15 for a proof, and see Algorithm 2.3 for the resulting algorithm.

To prove the correctness of recurrence (2.8), recall that we are considering additional peaks only. We first note that we can concentrate on the case $Q'[i, j] \neq -\infty$; this follows analogously to the proof for matrix Q and recurrence (2.5) in Sec. 2.5.1. So, assume $Q'[i, j] \neq -\infty$, which implies that there is a prefix path to x_i and a suffix path to x_j. These paths do not have to form a valid pair; but at least, they exist. For $i \neq j$, the argument that recurrence (2.8) is correct, is exactly the same as for the matrix Q. So, let us concentrate on the final case $i = j$: Assume that $x_L x_i \in E$ is the final edge of the prefix path, and that $x_{L'} x_j$ is the final edge of the suffix path. As we do not want to count the peak $x_i = x_j$ twice, we infer

$Q'[i,j] = Q'[L,j] = Q'[i,L']$. Again by an analogous argument as in the proof of recurrence (2.5), we see that $Q'[L,j] = \max_{l=0,\ldots,i-1}\{Q'[l,j] : x_l x_i \in E\}$ and $Q'[i,L'] = \max_{l=0,\ldots,j-1}\{Q'[i,l] : y_j y_l \in E\}$, which concludes the proof. □

What about the generalizations of (2.8) to the cases where peaks are missing, the scoring is no longer symmetric, or we even use an arbitrarily edge-weighted spectrum graph? An unsymmetrical scoring is easily dealt with; just include both sides of (2.9) in recurrence (2.8) for the case $i = j$. But things are slightly more complicated: The fact that we have pushed the weight of a node to all incoming edges, breaks the symmetry of the problem. Still, I believe that one can come up with a recurrence, although I expect it to be more complicated than the ones presented in this chapter. But the conceptually simpler solution is to remember that we originally wanted to score nodes (peaks), not edges; see Exercise 2.12. To this end, we can define the weight of a path to be the sum of node weights; and, we can define the *valid* weight of a path as the sum of weights where, if x_i and y_i are present simultaneously, only the larger weight is added to the weight of the path. See Exercise 2.17 for details.

2.6 Ion series: The abc and xyz of peptide fragmentation

If you take a look at any real-world peptide fragmentation spectrum, there is obviously more going on than what we pretended above. Compare Fig. 2.7 to Fig. 2.1: Besides the two "main" ion series b and y, there are at least four more ion series, namely a, c, x, and z ions. The following description is tailored toward CID (Collision-Induced Dissociation) peptide fragmentation, which still is the predominant method for this purpose.

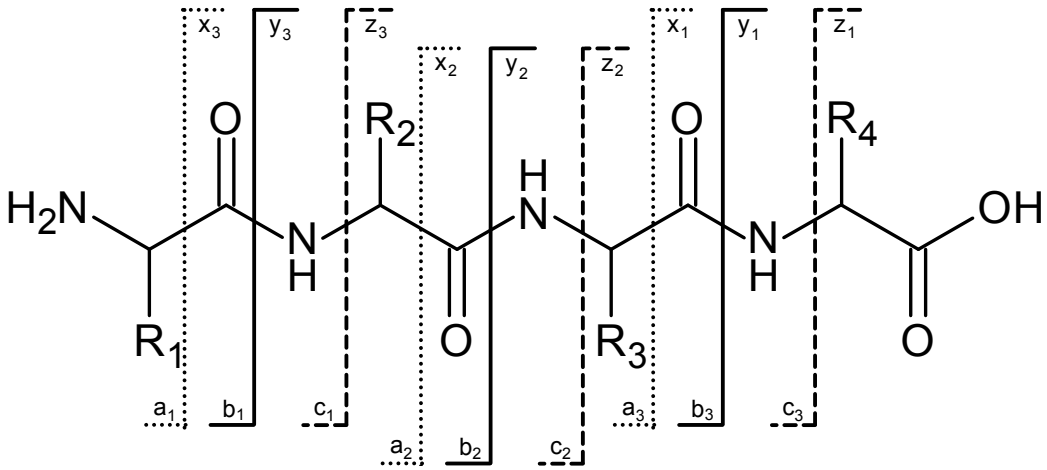


Figure 2.7: Fragmentation of a peptide into a,b,c and x,y,z-ions. This figure oversimplifies the fragmentation process.

Molecular formula modifications for six ion series are shown in Table 2.2. The a, b, and c ion series correspond to prefixes of the peptide; the x, y, and z ion series correspond to suffixes. To calculate the molecular formula of an ion from a prefix or suffix string, add up the molecular formulas of residues from Table 2.1 on page 18; then, add or subtract the molecular formula modification from Table 2.2, plus H^+ for a single charge. Note that I have omitted the proton from the molecular formula modifications in Table 2.2, to conform with the rest of the book. Just as b and y ions are complementary, the same holds for a and x ions, and for c and z ions. The molecular formula of a pair "b plus y" and a pair "c plus z" adds up to the molecular formula of the peptide. This is not the case for a pair "a plus x", whose molecular formulas add up to the peptide molecular formula minus H_2. The reason is that peptide fragmentation is still more complicated than Fig. 2.7 suggests: It is an involved process that can comprise a series of rearrangements inside the peptide, before the actual fragmentation takes place. We will not go into further details.

prefixes, N-terminal			suffixes, C-terminal		
series	MFM	mass	series	MFM	mass
a ions	$-CO$	−27.994915	x ions	$+CO_2$	+43.989829
b ions	none	±0.0	y ions	$+H_2O$	+18.010565
c ions	$+NH_3$	+17.026549	z ions	$-NH_3 + H_2O$	+0.984016

Table 2.2: Ion series for peptide fragmentation. 'MFM' is the molecular formula modification that has to be applied to the molecular formula of the prefix or suffix (that is, the sum of residue molecular formulas) to obtain the molecular formula of the ion series. Mass modification for protonation *excluded* from the table.

Example 2.3. Given the peptide ESI with molecular formula $C_{14}H_{25}N_3O_7$, assume that ES is the prefix (N-terminal fragment) and I is the suffix (C-terminal fragment). The molecular formula of the residue string ES is $C_5H_7NO_3 + C_3H_5NO_2 = C_8H_{12}N_2O_5$, whereas the residue string I has molecular formula $C_6H_{11}NO$. The full peptide has molecular formula

$$C_8H_{12}N_2O_5 + C_6H_{11}NO + H_2O = C_{14}H_{25}N_3O_7.$$

We calculate the corresponding molecular formulas of the ion series as:

series	molecular formula calculation		ion
a ion	$C_8H_{12}N_2O_5 - CO$	$= C_7H_{12}N_2O_4$	$C_7H_{13}N_2O_4^+$
b ion	$C_8H_{12}N_2O_5$	$= C_8H_{12}N_2O_5$	$C_8H_{13}N_2O_5^+$
c ion	$C_8H_{12}N_2O_5 + NH_3$	$= C_8H_{15}N_3O_5$	$C_8H_{16}N_3O_5^+$
x ion	$C_6H_{11}NO + CO_2$	$= C_7H_{11}NO_3$	$C_7H_{12}NO_3^+$
y ion	$C_6H_{11}NO + H_2O$	$= C_6H_{13}NO_2$	$C_6H_{14}NO_2^+$
z ion	$C_6H_{11}NO - NH_3 + H_2O$	$= C_6H_{10}O_2$	$C_6H_{11}O_2^+$

In fact, things are even more complicated: Ions of all ion series may also lose ammonia $-NH_3$ or water $-H_2O$. Unfortunately, y ions losing ammonia have the same molecular formula as z ions, so peaks are indistinguishable and intensities will add up in the mass spectrum. In practice, z-type ions are mostly observed after electron-based fragmentation as radical $z^\bullet$ ions, which are one hydrogen atom heavier. Next, immonium ions are "indicator ions" which do not correspond to a substring of the peptide string, but rather a single character. The molecular formula of an immonium ion is the molecular formula of the amino acid residue, minus CO, plus H^+. Immonium ions cannot be used for determining the sequence of the peptide, but they are indicative of the presence or absence of a particular amino acid from the sequence. Next, leucine and isoleucine are not as indistinguishable as we have thought: They sometimes may be told apart by different "fragmentation behavior". Next, internal fragments can occur, which are neither prefixes nor suffixes but rather substrings of the peptide string. Next, the spectrum may contain multiply charged fragments, see Section 1.4.2. And there is more; but these details are beyond the scope of this textbook.

2.7 Post-translational modifications: Enlarging the alphabet

When a proteomics expert takes a look at Table 2.1, he or she might object, "and where are the amino acid modifications?" We will cover them now, for the sake of completeness. In fact, we can cover all of these modifications without *any* changes to our approach. This is fundamentally different from peptide database searching (see Chapter 4 below) where variable modifications pose a major combinatorial problem. But the fact that our computational *de novo* sequencing approach does not require any changes, does not mean that modifications are easy to deal with: In fact, *de novo* sequencing becomes considerably

harder when variable modifications (see below) are present, as it further increases the ambiguity of the data.

We have to differentiate between two types of modifications of amino acids: The first is due to the experimental setup, such that all amino acids of a certain type are replaced by their modified counterpart. This is called a *fixed modification*. The most common example is carboxamidomethyl cysteine (CamC), where cysteine reacts with iodoacetamide. To deal with this situation, we simply replace the letter C in our alphabet, which now has molecular formula $C_5H_8N_2O_2S$ instead of C_3H_5NOS. Another example is the oxidation of methionine, which happens spontaneously during the analysis; so, experimentalists sometimes make sure that *all* methionine in the sample is oxidized. If we treat methionine as a fixed modification, since we carried out our experiments accordingly, then we can simply replace the letter M in our alphabet, which now has molecular formula $C_5H_9NO_2S$ for methionine sulfoxide instead of C_5H_9NOS. Fixed modifications make peptide *de novo* sequencing neither simpler nor more complicated: We simply replace one molecular formula of the character by a different one.[4] We do not introduce a new symbol for the modified amino acids, as they are an artifact of the experimental setup, and have no biological meaning.

The second type of modifications are *variable* modifications, enlarging the alphabet of amino acids that we have to look at: *Post-Translational Modifications* (PTMs) are chemical modifications of a protein after its translation. One of the most common PTMs is the phosphorylation of serine, threonine, and tyrosine: Phosphorylation is the addition of a phosphate group to a protein, and activates or deactivates many protein enzymes. It results in a molecular formula change of $+HPO_3$ for the affected amino acid residue. Any serine, threonine, tyrosine amino acid of the protein can or cannot be phosphorylated individually. This results in three *additional* amino acid residues that we have to take into account: For example, we introduce a new letter "pS" for the phosphorylated serine residue with molecular formula $C_3H_6NO_5P$. Other common post-translational modifications are deamidation of glutamine (Q) or asparagine (N); and carboxylation of aspartic acid (D) or glutamic acid (E). We may also include the methylated form of some amino acids, such as methylated arginine (R^*) with molecular formula $C_7H_{14}N_4O$, and doubly methylated arginine (R^{**}) with molecular formula $C_8H_{16}N_4O$.

Clearly, this makes the *de novo* sequencing problem more challenging, as the number of ambiguities increases through the additional characters we can add at every position. Surprisingly, the number of candidates we have to consider may *per se* not increase "dramatically": If we consider an alphabet of 25 instead of 19 letters, then the number of peptides of length exactly 10 increases from $6.13 \cdot 10^{12}$ (trillions of candidates) to $9.54 \cdot 10^{13}$ (tens of trillions). I would argue that it depends on the actual weighted characters we are adding: If we add characters that generate additional "combinatorial pitfalls" such as $\mu(\mathrm{AD}) = \mu(\mathrm{EG})$ (see Section 8.6 for details) then *de novo* sequencing may indeed become substantially harder, even if few characters are added.

Another common post-translational modification is glycosylation, the covalent attachment of oligosaccharides to the protein. As oligosaccharides are themselves polymers, these modifications can be very complex. We will come back to oligosaccharides in Chapter 11.

2.8 A two-step strategy for *de novo* sequencing

In many cases, our *de novo* sequencing algorithm will not be smart enough to identify the correct peptide string. But the algorithm was designed with the main objective of being *swift*: We have to consider, say, $3.76 \cdot 10^{25}$ candidate peptide sequences of length 20. A highly accurate yet slow algorithm which, for example, considers each candidate individually, is of no use in practice as it will run for billions of years.

But maybe, we are judging the algorithm too sternly? Maybe, we do not have to find "the" correct peptide sequence; maybe, it is sufficient if the algorithm returns several answers, and the correct answer

[4] In theory, it is possible that the modified mass equals that of another amino acid by chance; or, that a modified mass does *no longer* equal that of another amino acid. In practice, this subtlety appears to be irrelevant.

is just one of them? We can indeed modify our algorithm to return more than one string; and this is indeed often sufficient for high-quality *de novo* sequencing. Let us start with the second, easier step.

Suppose our algorithm has "sequenced" k candidates, one of which is (hopefully) the correct answer. We can now regard these candidates as a "custom database" we want to search in, and we can use methods for database searching (Chapter 4) to establish which of the candidates is the best match for the data. Since the number of candidates (say, a thousand strings) is much, much smaller than the total number of strings we have to consider in *de novo* sequencing, this means that we can now spend substantial time to score each candidate. This allows us to use much more involved scorings which can also take into account peak intensities, infrequent ion series, internal fragments or multiply charged fragments.

What about the first step? As we have used dynamic programming to derive the optimal solution, it is actually simple to also find suboptimal solutions. In practice, it is not advisable to fix a particular number k and ask for the k best solutions, see for example the problem of computing k-shortest paths [104]. But our choice of k is somewhat arbitrary, anyway; why not $2k$ or $0.7k + 23$? Instead, we will concentrate on computing suboptimal solutions that deviate only by a small margin from the score of the optimal solution. First, we compute the exact score T of an optimal solution. Assume we are given some $\varepsilon > 0$, and we want to find all strings with score at least $T - \delta$ where $\delta := \varepsilon \cdot T$. This can be done using backtracking, allowing suboptimal scores in the recursion step: We simply keep track of how much we deviate from the optimal solution in the current step. Let the "deviation from optimal" of the partial solution constructed so far be d; we only further process those partial solutions where $d \leq \delta$.

To generate a candidate set of reasonable size for the subsequent analysis, we can iteratively adjust δ: Assume that we want to generate at least l and at most u solutions. We start with a reasonable "guessed" δ. We stop the recursion if we have already generated more than u solutions; in this case, we decrease δ (say, by interval-halving) and restart. If we have not generated enough solutions, we increase δ and restart; we either double δ, or use interval-halving if we have previously established an upper bound.

Suboptimal solutions can be rather uninteresting minor modifications of the optimum solution; for *de novo* sequencing, the algorithm can decide to fill larger gaps in the sequence with all strings that have the same mass as the gap. In practice, you may have to prohibit this behavior, to avoid having to score a large number of peptide strings which are basically identical.

Some published approaches for peptide *de novo* sequencing make use of this two-step approach; for example, PEAKS [226] computes the 10 000 best-scoring candidates and re-evaluates them using a more stringent scoring. (Novor [223] uses a two-step approach, but the second step is only refining ambiguities in the original sequence.) For related problems such as glycan "sequencing" (Chapter 11), this approach is also used.

2.9 Shotgun proteomics: Shoot first, ask later

Now that we have spent some time on the problem of peptide sequencing, the question is: Why is this of interest? Today, most proteomics experiments rely on shotgun proteomics using *Liquid Chromatography Mass Spectrometry* (LC-MS), see Sec. 1.6.2. The basic idea is to digest the proteins in the mixture, and only afterward to separate the peptides using liquid chromatography. In comparison to the alternative workflow of separating proteins first, then digesting them individually (for example, for *peptide mass fingerprinting*), this experimental setup has the huge advantage that it requires much less work experimentally: Separation is performed fully automatically by the instrument.

The name "shotgun proteomics" is somewhat misleading: Instead of dumb pellets, we are firing with smart missiles that find their targets (lysine and arginine) with high accuracy. The "shotgun" part is rather when we try to make sense of the peptide identifications since, after all, we are interested in proteins.

By design, only peptides are identified in shotgun proteomics. Now, different proteins may "generate" the same peptide through tryptic digestion, which is very common for protein isoforms and homologous proteins; in addition, some of our peptide identifications (sequences) are wrong or ambiguous. Hence, the last step of the shotgun proteomics identification pipeline is mapping peptides to proteins. (This step

requires a protein database, so this is relevant rather for database searching than *de novo* sequencing.) This is usually done by statistical inference algorithms [321] which provide us, for each protein, with a probability that it is present in the sample. Unfortunately, this step is currently not covered in this textbook, see Chapter 13.

2.10 Historical notes and further reading

The title of Sec. 2.6 was borrowed from the paper by Steen and Mann [334], see there for more details on peptide *de novo* sequencing. Mass spectrometry experts still sequence peptides "by hand", see Seidler, Zinn, Boehm and Lehmann [318] for a review. The nomenclature of ion series is due to Roepstorff and Fohlman [299]. Zhang [377] quantitatively modeled the peptide fragmentation process.

There exists a huge number of computational approaches for *de novo* sequencing of peptides, see Muth *et al.* [247] for a review. Early approaches [305] were based on exhaustive enumeration of all peptide strings and, hence, limited to very short peptides. Pruning techniques were developed to reduce the combinatorial explosion of the problem [143, 370] but did not prove very successful, in particular because a correct sequence prefix could be pruned due to peaks missing in the measured spectrum. Some noteworthy approaches are: Lutefisk [344] from 1997, PEAKS [226] (which is now commercial), PepNovo [116], and Novor [223] by Bin Ma, the creator of PEAKS. See Muth and Renard [248] for a more complete list. Muth and Renard [248] also compared the performance of Novor, the commercial PEAKS software, and PepNovo, and investigated the most common sequencing errors. Tran *et al.* [349] introduced a *de novo* sequencing method based on deep learning which, according to the authors, performs better than Novor, PEAKS or PepNovo. Since then, numerous deep learning methods have followed, see Bittremieux *et al.* [23] for a review.

Our presentation of this chapter loosely follows the paper of Chen, Kao, Tepel, Rush and Church [56], with major modifications to simplify the line of thought. The algorithm of Sec. 2.5.4 (a proper prefix mass may equal a proper suffix mass) is not in this paper. There is a reasonable number of peptides with b and y ions of identical mass, see Exercises 3.13 and 3.14 below; so, this is a relevant generalization.

It is common in computational graph theory to search for longest paths in edge-weighted rather than in node-weighted graphs. To this end, both our presentation (except for Sec. 2.5.4) as well as the literature [12, 55, 56, 70] use the trick of transforming node weights into edge weights. In fact, there is a good reason for edge-weighting the graph that stems from the application itself: In this way, we can also score the mass difference between consecutive peaks of a peak series, as well as the existence or non-existence of such consecutive peaks.

The spectrum graph was introduced by Bartels [12] in 1990. Valid paths in spectrum graphs are a particular case of antisymmetric paths. When Dančík *et al.* [70] cast the peptide *de novo* problem onto the Antisymmetric Longest Path problem they noted that, in general, this is an NP-hard problem [118]. But the authors already conjectured that, due to the special structure of the spectrum graph, the *de novo* sequencing problem may allow for a polynomial-time algorithm. As we know, such an algorithm was found only a year later [55]. Andreotti *et al.* [6] presented a more flexible method for finding longest antisymmetric paths, which is much faster than solving the integer linear program directly. It is based on Lagrangian relaxation of the integer linear program, which pushes the "NP-hard part" of the problem into the objective function; in practice, this also boils down to repeatedly solving the longest path problem in a directed acyclic graph using dynamic programming.

Searching for antisymmetric longest paths inside the spectrum graph has certain intrinsic shortcomings: First, we can only "see" the mass difference between two peaks, but not the peak masses themselves. In this way, small errors can add up to larger errors of prefixes or suffixes, which are far beyond the mass accuracy we would find acceptable (Exercise 2.18). Second, it is non-trivial to integrate other ion types into our recurrence [10]. Third, searching for antisymmetric paths is slow (two-dimensional DP) and not necessary in practice: As Mo *et al.* [242] suggested, we can search for the optimal peptide string, accepting that certain peaks are counted twice, as prefix and suffix mass. We then identify these peaks; assuming

there is only a small number of them, we fix these peaks to be prefix or suffix mass, and rerun the algorithm multiple times. In practice, the number of peaks that are counted twice is small, so this approach is very fast for real-world data. The algorithmically more elegant solution to this problem would be again Lagrangian relaxation; unfortunately, the formulation of Andreotti *et al.* [6] does not prevent peak double counting but rather forbids prefixes and suffixes of identical mass, as in our original, idealized formulation of the problem.

So, if "double peak counting" can be avoided efficiently by this heuristic, then why did we spend a complete chapter on the exact solution? There are several answers to that: First, historical reasons: This is one of the first (if not *the* first) algorithms ever published in computational mass spectrometry that fulfills our strict criteria from the preface. Second, it does not matter: The computational solution is very instructive, and we will reuse ideas from this chapter throughout the textbook. Third, didactic reasons: I teach the Smith-Waterman algorithm before I teach BLAST.

If you are wondering why we paid so much attention to the overly simplified peak counting score, you might want to "sneak preview" glycan *de novo* sequencing in Chapter 11: It turns out that this is a computationally hard problem even for the peak counting score.

2.11 Exercises

2.1 Assume that our tandem MS spectrum was solely made up of y ions, corresponding to suffix masses. Then, an interpretation of the spectrum would be much easier. Describe an algorithm that, given a spectrum $\mathcal{M} = \{m_1, \ldots, m_n\}$ with $m_1 < m_2 < \ldots < m_n$ and precursor mass $M = m_n$, reconstructs the peptide string from the spectrum. What is the time complexity of your algorithm?

2.2 Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ be a weighted alphabet with $\mu(\mathsf{a}) = 2$, $\mu(\mathsf{b}) = 3$, $\mu(\mathsf{c}) = 7$, and $\mu(\mathsf{d}) = 10$. Find all strings that have the same fragmentation spectrum as aabdac. Explain why there are no other strings.

2.3 With Σ from the previous exercise, find a string s of length $|s| \geq 2$ that has a *unique* fragmentation spectrum; that is, there is no other string $s' \in \Sigma^*$ with $\mathcal{M}(s) = \mathcal{M}(s')$.

2.4 For Σ from Exercise 2.2, find a string that generates the fragmentation spectrum $\mathcal{M} = \{0, 2, 3, 5, 9, 11, 12, 14\}$, where the precursor mass is $M = 14$. Note that there are several such strings; can you find them all?

2.5 Develop a branch-and-bound algorithm for finding all strings $s \in \Sigma^*$ with $\mathcal{M}(s) = \mathcal{M}$ for a given set of masses $\mathcal{M}$. Your algorithm should build up prefixes of the string, then recurse for each character that can be appended.

★ 2.6 Modify the algorithm of Sec. 2.4 for *ideal data* so that it uses only linear memory. To this end, strip off those parts of the DP matrix D that are "uninteresting". Show how to backtrace through this reduced matrix.

2.7 Instead of explicitly building the spectrum graph, it is sufficient to keep an implicit representation of its edge set. Explain how this can be done for ideal data.

2.8 We are given a tandem MS spectrum $\mathcal{M} = \{m_1, \ldots, m_n\}$. We assume that this spectrum consists solely of prefix masses and noise peaks, so no suffix masses are present. Here, some string s explains a mass $m \in \mathcal{M}$ if s has a prefix of mass m. Describe an algorithm that finds a string $s \in \Sigma^*$ maximizing the number of explained masses. Show that your algorithm has running time $O(n\,|\Sigma|)$ or, if we assume the alphabet to be constant, time $O(n)$.

2.9 Assume that there are additional peaks but no missing peaks, as introduced in Sec. 2.5.1. Prove that the maximum number of peaks in a mass spectrum that can be explained by any string equals $2 \max_{i,j}\{Q[i,j] : x_i y_j \in E\}$, using the definition of Q and w from that section.

★ 2.10 Prove Lemma 2.2.

2.11 Given the weighted alphabet $\Sigma = \{\mathsf{a},\mathsf{b},\mathsf{c}\}$ with $\mu(\mathsf{a}) = 2$, $\mu(\mathsf{b}) = 3$, and $\mu(\mathsf{c}) = 7$, and a tandem MS spectrum $\mathcal{M} := \{0, 2, 7, 8, 9, 14, 16, 17, 22, 24\}$ with precursor mass 24. We know that some of the peptide peaks might be missing, and that some of the measured peaks might be noise. Find a string that explains a maximum number of peaks.

2.12 We are given a set of masses $\mathcal{M}$ with $0, M \in \mathcal{M}$, such that $m \in \mathcal{M}$ implies $M - m \in \mathcal{M}$ for precursor mass M. In addition, we are given a prefix score $w_1 : \mathcal{M} \to \mathbb{R}$ and a suffix score $w_2 : \mathcal{M} \to \mathbb{R}$: Here, $w_1(m)$ is added to the score of a string if the string has a proper prefix of mass m, and $w_2(m)$ is added if the string has a proper suffix of mass m. Formally, we define

$$\text{score}(s) := \sum_{\text{proper prefix } a \text{ of } s} w_1(\mu(a)) \; + \sum_{\text{proper suffix } b \text{ of } s} w_2(\mu(b))$$

Show how to compute the optimal solution using recurrence (2.5).

2.13 Consider the weighted alphabet from Example 2.1 and the score from Sec. 2.5.4 that counts proper prefix masses plus proper suffix masses present in the spectrum. Find a string s whose proper prefix masses and proper suffix masses are all different, and a string s' with $\mu(s') = \mu(s)$ that receives a higher score than s for the ideal spectrum $\mathcal{M}(s)$.

2.14 Let the score of a string be the number of explained peaks in the measured spectrum, minus the number of missing peak pairs: This is the number of prefix/suffix peak pairs in $\mathcal{M}(s)$ that are not present in the measured spectrum $\mathcal{M}$. Show that recurrence (2.5) with weighting w from (2.6), modified by (2.7), will compute the optimal solution.

2.15 Show that

$$\max_{l=0,\dots,i-1} \{Q'[l,j] \,:\, x_l x_i \in E\} = \max_{l=0,\dots,j-1} \{Q'[i,l] \,:\, y_j y_l \in E\}$$

holds in recurrence (2.8) for the case $i = j$.

2.16 With the weighted alphabet from Example 2.1, we have measured a tandem mass spectrum

$$\mathcal{M} = \{0, 2, 8, 9, 11, 12, 14, 15, 21, 23\}$$

with precursor mass $M = 23$. Assume that there are "additional peaks only". Find the string that explains a maximum number of peaks, using recurrence (2.8) and matrix Q', as the true solution may contain prefix peaks and suffix peaks of identical mass.

★ 2.17 Let $G = (V, E)$ be a spectrum graph for some set of masses $\mathcal{M} = \{x_0, \dots, x_n, y_n, \dots, y_0\}$ with $x_i + y_i = M$ for all $i = 0, \dots, n$. Let $w : V \to \mathbb{R}$ be arbitrary *node weights*. We define the *valid length* of a path to be the sum over all node weights where, if x_i and y_i are simultaneously present in the path, we add the maximum weight of x_i or y_i (but not both). Define a matrix Q' and find a recurrence analogous to (2.8) that can be used to compute the maximum valid length of any path in G.

2.18 Find a series of b ion peak masses where, for mass error $\varepsilon = 0.5$ Da, the mass difference between any two consecutive peaks can be explained by the mass of an amino acid residue, but the mass of the last peak cannot be explained by a peptide b ion.

★ 2.19 Assume that the unknown peptide contains exactly one Post-Translational Modification (PTM) but unfortunately, we do not know the mass of the modified amino acid. We assume that we have ideal data. Reconstruct the peptide strings and the mass of the PTM amino acid from the measured set of masses $\mathcal{M}$ using recurrence (2.3), plus a modified version of it. The trick is to build a matrix similar to D but this time, from the "center peaks" x_n, y_n outward.

2.20 For ideal data, $\mathcal{M}(s) = \mathcal{M}(s^{-1})$, so we cannot tell apart a peptide and its inverse. Why is this not an issue in practice?

★ 2.21 Assume there are several peptides with similar precursor mass (say, no more than 25 Da between any two peptides) with experimental fragmentation spectra $\mathcal{M}_1, \ldots, \mathcal{M}_k$. All we are given is the spectrum $\mathcal{M} := \mathcal{M}_1 \cup \cdots \cup \mathcal{M}_k$; we do not even know k. Develop a computational approach that determines high-quality substrings of length 5 or 6 for each of the unknown peptide strings.

3 Combinatorics of Weighted Strings

"If numbers aren't beautiful, I don't know what is." (Paul Erdős)

We now turn to some problems that keep reappearing, in various flavors, not only throughout this textbook but also throughout the computational mass spectrometry literature. These problems, and also our strategies for solving them, lie at the core of many MS applications. In fact, we have already stumbled upon some of the problems in the previous chapter. All problems circle around the question of decomposing masses: We are given a weighted alphabet, such as the alphabet of amino acid residues; and we want to know if and how peak masses that we see in a mass spectrum can be explained.

For simplicity, we assume throughout this chapter that all masses are integers: For example, we can round amino acid masses to the closest integer. Alternatively, we multiply all masses by a large constant c such as $c = 1000$ before rounding, to reduce the impact of rounding errors. See Sec. 8.1 below for a thorough investigation of how to decompose real numbers, and how this can be applied in metabolomics.

3.1 Formal problem definitions

We are given an alphabet $\Sigma = \{a_1, \ldots, a_k\}$, for example the alphabet of amino acids, or an alphabet of elements. Throughout this chapter, we denote the cardinality of our alphabet by $k = |\Sigma|$. We are also given a mass function $\mu : \Sigma \to \mathbb{N}$. Recall that the mass of a string $s = s_1 \ldots s_n$ over Σ is defined as $\mu(s) := \sum_{i=1}^{n} \mu(s_i)$. In the following, we usually assume that all characters have pairwise different mass, even though this is not required for some of the algorithms. Still and all, there are very few applications where different characters of the same mass are reasonable: Instead, the method of choice usually is to treat all characters of identical mass as one, and to sort out this impreciseness at a later stage, see Exercise 3.6. Recall that for the amino acid alphabet, leucine and isoleucine have identical mass and will be regarded as one character.

One should immediately notice that the order of characters in the string has no effect on the mass: only the number of occurrences of each of the characters is important. To this end, we make the following definition: A *compomer* over Σ can be viewed either as a map $c : \Sigma \to \mathbb{N}$, or as a vector $(c_1, \ldots, c_k) \in \mathbb{N}^k$.[1] Defining compomers as vectors is easier to grasp, whereas defining them as maps is mathematically more elegant: We do not have to order the alphabet, and the definition also works for infinite alphabets. In the following, we will use these two definitions interchangeably. If we view compomers as vectors $c = (c_1, \ldots, c_k)$, the order of characters in the alphabet is relevant, and we assume this order to be arbitrarily fixed. Given a string $s = s_1 \ldots s_n$, the function $\mathrm{comp} : \Sigma^* \to \mathbb{N}^k$ maps s to its compomer by counting characters,

$$\mathrm{comp}(s) = (c_1, \ldots, c_k) \quad \text{with} \quad c_j = \#\{i : s_i = a_j\}. \tag{3.1}$$

The *length* of compomer c is $|c| := \sum_{j=1}^{k} c_j$, and the *mass* of c is $\mu(c) := \sum_{j=1}^{k} c_j \cdot \mu(a_j)$. The definition of length of a compomer, becomes obvious by the following lemma:

Lemma 3.1. *Given a string $s \in \Sigma^*$ and a compomer $c := \mathrm{comp}(s)$. Then $|c| = |s|$ and $\mu(c) = \mu(s)$.*

We will often denote a compomer c as $(a_1)_{c_1} \ldots (a_k)_{c_k}$, omitting those characters a_i with $c_i = 0$. This increases readability and is particularly favorable for large alphabets, such as amino acids. This presentation is obviously inspired by molecular formulas from chemistry, such as $C_{12}H_{22}O_{11}$ for sucrose. This shows that compomers also exist "without strings", see Chapter 7. Sometimes, compomers with

[1] Compomers have been proposed numerous times throughout the literature, and many different names have been proposed such as *compositions* [19], *Parikh vectors* [306], *multiplicity vectors* [8], or *abelian patterns*.

negative entries make sense: For example, the difference between two compomers can be useful in certain applications.

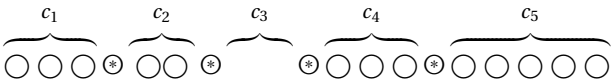
Example 3.1. Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ be our alphabet with masses $\mu(\mathsf{a}) = 2$, $\mu(\mathsf{b}) = 3$, $\mu(\mathsf{c}) = 7$, and $\mu(\mathsf{d}) = 10$. We will make use of this weighted alphabet throughout this chapter. For a string $s = \mathsf{baacbcaca}$ we have $c := \mathrm{comp}(s) = (4, 2, 3, 0)$ in vector notation or, equivalently, $c = \mathsf{a}_4\mathsf{b}_2\mathsf{c}_3$. Now, $|c| = 9$ and $\mu(c) = 4 \cdot 2 + 2 \cdot 3 + 3 \cdot 7 = 35$.

Now, we turn to an obvious question: It is well known that there are k^n strings of length n over an alphabet of size k. Now, how many compomers of a given length exist?

Lemma 3.2. *The number of compomers of length n over an alphabet of size k is $\binom{n+k-1}{k-1}$.*

Hence, the rate of growth is polynomial in n, not exponential. But for large alphabets, the number is still increasing rapidly: For a fixed alphabet of size k, we have $\binom{n+k-1}{k-1} \in \Theta(n^{k-1})$ many compomers. For the amino acid alphabet of size 19, this implies that the number of compomers is increasing with a polynomial of degree 18.

Lemma 3.2. Every compomer of length n can be mapped bijectively onto $n + k - 1$ points, where $k - 1$ points are selected by asterisks. The compomer $(c_1, \ldots, c_k)$ is mapped to c_1 points, asterisk, c_2 points, asterisk, ..., asterisk, c_k points. For example, $(c_1, \ldots, c_5) = (3, 2, 0, 3, 5)$ is mapped to:



How many possibilities exist to choose (cross out) $k - 1$ points out of $n + k - 1$ points? It is well known that there exist $\binom{n+k-1}{k-1}$ such possibilities. □

There are four problems that we will address in the following: Given an integer mass $M \geq 0$, what is the number of compomers and strings with this mass? Is there at least one such compomer or string? If yes, can we provide a witness or proof, that is, a compomer c with $\mu(c) = M$? And finally, can we enumerate all compomers of mass M? Searching for compomers or strings over an alphabet Σ with mass M, we also say that we *decompose* mass M, and the compomers or strings of mass M will be called *decompositions*.

In combinatorics, determining the number of things is called "counting", whereas "enumerating" refers to constructing all objects with a particular property.[2] Usually, counting can be achieved faster than enumerating. In turn, counting (how many?) is at least as hard as the decision problem (at least one?).

In the remainder of the chapter, we name the characters of the alphabet with their integer masses: Instead of the alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ from Example 3.1, we will consider the alphabet $\Sigma = \{2, 3, 7, 10\}$. This will make it easier to follow the formalism. As we assume that all characters have pairwise distinct masses, this is not a restriction. Unless explicitly stated otherwise, we assume that *all masses are positive*. Finally, let us assume that masses in the alphabet $\Sigma = \{a_1, \ldots, a_k\}$ are ordered, so in particular, a_1 is the smallest mass and a_k is the largest mass.

Finally, let us consider the problem of negative integer masses. If all masses are negative, you can take the (additive) inverse of all masses, and you are back at our previous problem. If at least one mass is positive and one mass is negative, then every mass that has a decomposition has an infinite number of them, see Exercise 3.22, so counting and enumerating do not make sense. But sometimes you are not interested in enumerating all solutions but only some optimum one following, say, the parsimony principle; for example, we might be interested in the minimum number of post-translational modifications that makes some protein fit with some peak mass.

[2] Depending on the scientific area and the year of publication, you sometimes find "to enumerate" as a synonym for "to count"; prominent examples include Garey and Johnson [118], or "graph enumeration" that deals with counting certain graphs. To reduce confusion, we will stick to this sharp differentiation throughout this textbook.

3.2 Counting compomers and strings

We have seen in Lemma 3.2 how to compute the number of compomers of given length. In mass spectrometry, the usually more interesting question is: How many compomers exist with mass M? We will present an exact solution based on dynamic programming (see Sec. 15.5) that is actually very simple — a related problem is "scientific folklore" in computer science and combinatorics, see Exercise 3.1. We solve the problem by two-dimensional dynamic programming. Let C be a two-dimensional table, where $C[i,m]$ is the number of compomers c over the alphabet $\{a_1,\dots,a_i\}$ with mass $\mu(c) = m$, for $i = 0,\dots,k$ and $m = 0,\dots,M$. For $i < k$, this means that we do not take into consideration the complete alphabet but only a sub-alphabet. In the extreme case $i = 0$, this corresponds to an empty alphabet and, obviously, the only mass that we can decompose over this alphabet is $m = 0$, and there is exactly one decomposition (the empty compomer) for this mass. Hence, we initialize our table by $C[0,0] = 1$ and $C[0,m] = 0$ for $m = 1,\dots,M$.

Let us assume that we have previously computed all entries $C[i',m']$ with $i' \le i$ and $m' \le m$, where $i' < i$ or $m' < m$ or both holds. To compute the number of compomers with mass m over the alphabet $\{a_1,\dots,a_i\}$ we sort these compomers into two buckets: One bucket contains those compomers where $c_i = 0$ holds, the other bucket contains those with $c_i \ge 1$. Clearly, the two buckets are disjoint, so we can add up these two numbers to reach the desired value. But we already know these values: The number of compomers that *do not* use letter a_i is exactly $C[i-1,m]$, the number of compomers for mass m over the alphabet $\{a_1,\dots,a_{i-1}\}$. And for the compomers that use letter a_i at least once, we can remove a single letter a_i and count compomers of mass $m - a_i$ that use a_i at least zero times: This number is stored in $C[i,m-a_i]$. Obviously, the latter number is only meaningful in case $m \ge a_i$. In total, we reach the recurrence:

$$C[i,m] = \begin{cases} C[i-1,m] + C[i,m-a_i] & \text{if } m \ge a_i \\ C[i-1,m] & \text{else} \end{cases} \tag{3.2}$$

At the end of our computation, $C[k,M]$ holds the desired number. We formalized the above argument in the proof of Lemma 3.3.

Example 3.2. Consider the weighted alphabet $\Sigma = \{2,3,7,10\}$ from Example 3.1. How many compomers exist with mass $M = 13$? Using (3.2) we compute the following table:

i	a_i	$M=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
0	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	1	0	1	0	1	0	1	0	1	0	1	0	1	0
2	3	1	0	1	1	1	1	2	1	2	2	2	2	3	2
3	7	1	0	1	1	1	1	2	2	2	3	3	3	4	4
4	10	1	0	1	1	1	1	2	2	2	3	4	3	5	**5**

For example, $C[2,12] = C[1,12] + C[2,9] = 1 + 2 = 3$. So, the number of compomers is $C[4,13] = 5$. Note that the above table tells us the number of compomers for *any* $m \le 13$.

An algorithm that applies recurrence (3.2) has running time $O(kM)$, and requires $O(kM)$ space to store table C. In complexity theory, this is called a *pseudo-polynomial* running time: The running time depends linearly (polynomially) on the integer M which is part of the input. Regarding memory consumption, slight improvements are possible: For the computation of entries $C[i,\cdot]$, only entries of type $C[i-1,\cdot]$ and $C[i,\cdot]$ are needed. Hence, we can calculate the table row by row, and "forget" each row after computation of the subsequent row has been finished. Then, memory requirements are reduced to $O(M)$. Doing so, we can no longer ask for the number of decompositions for some sub-alphabet $\{a_1,\dots,a_i\}$ for $i < k$, but this is of minor concern here.

In fact, a single array $C[0\dots M]$ is sufficient, which we overwrite row by row. Assume that the array holds row $i-1$, that is, $C[m] = C[i-1,m]$ for all m. We now go through the array from left to right, in the order of increasing mass $m = a_i,\dots,M$, and replace $C[m]$ by $C[m] + C[m-a_i]$. Why does this compute

Claude helped!

```
1:  function ComputeNumberCompomers(weighted alphabet Σ, mass M)
2:      array C[0...M] of integers
3:      C[0] ← 1; C[m] ← 0 for m = 1,...,M
4:      for i = 1,...,k do
5:          for m = a_i,...,M do
6:              C[m] ← C[m] + C[m − a_i]
7:          end for
8:      end for
9:      return array C
10: end function
```

Algorithm 3.1: Computing the number of compomers over an alphabet $\Sigma = \{a_1, \ldots, a_k\}$ of integer masses, up to some maximum mass M.

row i? When we reach position m, the entry $C[m]$ has not been touched yet, so it still holds the old value $C[i-1, m]$; this is exactly the first summand of (3.2). Position $m - a_i$ lies to the left of m, so it already holds the new value $C[i, m - a_i]$; this is exactly the second summand. Positions $m' < a_i$ are never overwritten, but that is fine, as $C[i, m'] = C[i-1, m']$ holds for them. For example, let the array hold row 1 of the table above, and let $a_2 = 3$: We compute $C[3] \leftarrow 0 + C[0] = 1$, $C[4] \leftarrow 1 + C[1] = 1$, $C[5] \leftarrow 0 + C[2] = 1$, and $C[6] \leftarrow 1 + C[3] = 2$, where $C[3]$ already holds the new value; this is $C[2, 6] = 2$ as in the table. Note that the order is crucial: If we went through the array from right to left, then $C[m - a_i]$ would still hold the old value $C[i - 1, m - a_i]$ when we update $C[m]$, and we would count only those compomers that contain character a_i at most once.

An implementation of this idea is given in Algorithm 3.1. If we are only interested in the number of compomers for mass M but not for any smaller masses, we can compute the table column by column, and reduce memory requirements to $O(k \max_i a_i)$. Note that the latter does not depend on M, which is very favorable in applications. On the other hand, we have to forget the number of decompositions for almost all $m < M$, which is unattractive in applications. We reach:

Lemma 3.3. *For an alphabet $\Sigma = \{a_1, \ldots, a_k\}$ of integer masses, the number of compomers with mass m, for each $m = 0, \ldots, M$, can be computed in $O(kM)$ time and with $O(M)$ space using Algorithm 3.1.*

The formal proof of this lemma can be found in the next section.

The related question for strings is, how many strings over Σ have mass M? This question is rather simple to answer, and requires only one-dimensional dynamic programming: Let $C'[m]$ denote the number of strings over Σ that have mass exactly m. Then, each such string can be divided into a string that is one character shorter, plus one character. Now, we can sort the strings into k many buckets, depending on the last character, and see that all of these buckets are disjoint. We initialize $C'[0] = 1$ as there is exactly one string (the empty string) of mass zero. We easily reach the recurrence:

$$C'[m] = \sum_{i=1}^{k} C'[m - a_i] \tag{3.3}$$

where we assume that $C'[m] = 0$ holds for all $m < 0$. Put differently: Iterate over all $i = 1, \ldots, k$ and for those that satisfy $m \geq a_i$, add $C'[m - a_i]$ to the total number of strings.

Example 3.3. Consider again the weighted alphabet $\Sigma = \{2, 3, 7, 10\}$. How many strings exist with mass $M = 13$? Using (3.3) we compute:

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$C'[m]$	1	0	1	1	1	2	2	4	4	7	10	12	20	**25**

Hence, the number of strings is $C'[13] = 25$. Even for this small example, we observe the different rate of growth for compomers vs. strings, namely, polynomial vs. exponential: Regarding $m = 5$ there exists only one compomer $\mathsf{a}_1\mathsf{b}_1$, but two strings ab and ba.

Note that the problem of counting weighted strings, has a striking similarity with Fibonacci numbers: In fact, for $\Sigma = \{1, 2\}$ we reach the definition of these numbers, $F(n) = F(n-1)+F(n-2)$.[3] For an arbitrary alphabet, (3.3) defines a linear recurrence relation with finite history and constant coefficients [136], or a *linear recursive sequence* for short. In theory, we can find a closed-form solution for any linear recursive sequence. Then, the exact number $C'[m]$ can be computed in constant time, just like the nth Fibonacci number $F(n)$ can be computed as

$$F(n) = \frac{1}{\sqrt{5}}\left(\varphi^n - (1-\varphi)^n\right) = \left\lfloor \tfrac{1}{\sqrt{5}}\varphi^n + \tfrac{1}{2} \right\rfloor \tag{3.4}$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio, and $\lfloor\cdot\rfloor$ denotes the floor function for rounding down. The problem is that there exists no simple way to find this closed form for an arbitrary alphabet Σ, and that the resulting formula will be rather complicated.

3.3 Formal proof of the counting lemma

We now give a formal proof of Lemma 3.3. Other lemmata and claims in this chapter can be proven similarly, so we will go through this formal exercise only once. Readers with no formal background in mathematics or computer science might want to trust me on the subject matter, and skip this section altogether.

I present this proof as an example of the requirements mentioned in the preface of this textbook; namely, specification of the input, generalizability of the method, correctness of the algorithm, and running time of the algorithm. We have clearly stated the input of the method — an alphabet of positive integer masses and a non-negative maximum mass M — and we have not stated any restrictions that the algorithm might work only for certain inputs, but choose to fail on others. Now, we will *prove* that the algorithm works correctly for all input, and we will also prove its running time.

First, we show that recurrence (3.2) computes all entries $C[i, m]$ according to the *definition* of the $C[i, m]$: By this definition, $C[i, m]$ is the number of compomers of mass m, over the alphabet $\{a_1, \ldots, a_i\}$. We do so by induction on i and m. As our induction start, we observe that only mass $m = 0$ can be decomposed over the empty alphabet, having a unique decomposition, so the $C[0, \cdot]$ are correctly initialized. Assume that $i \geq 1$. Let C be the set of compomers over the alphabet $\{a_1, \ldots, a_i\}$ with $\mu(c) = m$ for all $c \in C$; then, $|C| = C[i, m]$ must hold. Partition C into two sets C_1, C_2 with $C_1 \cup C_2 = C$ and $C_1 \cap C_2 = \emptyset$:

$$C_1 := \{c \,:\, c = (c_1, \ldots, c_i) \in C, c_i = 0\}$$
$$C_2 := \{c \,:\, c = (c_1, \ldots, c_i) \in C, c_i \geq 1\}$$

Let C_1' be the set of compomers over the alphabet $\{a_1, \ldots, a_{i-1}\}$ of mass m; by induction, $\left|C_1'\right| = C[i-1, m]$ must hold. We can easily define a bijection between the sets C_1 and C_1', either removing the trailing zero, or appending it. Hence, $|C_1| = \left|C_1'\right| = C[i-1, m]$.

For $m < a_i$ we obviously have $C_2 = \emptyset$ and, hence, $C = C_1$, so our claim follows. Assume $m \geq a_i$: Then, let C_2' be the set of compomers over the alphabet $\{a_1, \ldots, a_i\}$ of mass $m-a_i$; by induction, $\left|C_2'\right| = C[i, m-a_i]$ must hold. We define a bijection $\varphi : C_2 \to C_2'$ by $\varphi(c_1, \ldots, c_{i-1}, c_i) := (c_1, \ldots, c_{i-1}, c_i - 1)$, removing one character a_i with mass a_i from the compomer. Hence, $|C_2| = \left|C_2'\right| = C[i, m-a_i]$. As C is the disjoint union of C_1, C_2 we reach

$$C[i, m] = |C| = |C_1 \cup C_2| = |C_1| + |C_2| = C[i-1, m] + C[i, m-a_i]$$

[3] Note that the sequence is shifted, though, as we set $C'[0] = 1$.

as claimed.

Next, we make sure that Algorithm 3.1 does in fact compute recurrence (3.2). We claim that at the start of the outer For-loop for some i, we have $C[m] = C[i-1,m]$ for all $m = 0,\dots,M$, where $C[i,m]$ is computed by (3.2). This clearly holds for $i = 1$ after initialization. Entries $C[m]$ with $m < a_i$ are not changed, and indeed $C[i,m] = C[i-1,m]$ for $m < a_i$. For $m \geq a_i$, the inner loop runs over increasing masses; when we update $C[m]$, it still holds $C[i-1,m]$, whereas $C[m-a_i]$ already holds $C[i,m-a_i]$, as it has either been updated before or not been changed at all. So, the update sets $C[m]$ to $C[i-1,m] + C[i,m-a_i] = C[i,m]$, and after the inner loop, we have $C[m] = C[i,m]$ for all m, as claimed. Claude helped!

Finally, we consider running time and memory of the algorithm: Space is clearly $O(M)$ for storing array C. Initialization requires $M+1$ assignments; afterwards, each of the k iterations of the outer loop requires at most M additions, so running time is $O(kM)$. □

3.4 Finding witnesses and the decision problem

We now turn to the slightly simpler question: Is there a compomer with mass M over the alphabet $\Sigma = \{a_1,\dots,a_k\}$? Note that we can answer this question by using our algorithms from the previous section, checking whether "$C[k,M] \geq 1$". We will now give a related solution but here, we only need a one-dimensional binary table A. We define $A[m] = 1$ if and only if there is at least one compomer of mass m over the alphabet $\{a_1,\dots,a_k\}$. We initialize $A[0] = 1$, and use the recurrence

$$A[m] = \begin{cases} 1 & \text{if there is some } i \text{ with } A[m-a_i] = 1 \text{ for } m \geq a_i, \\ 0 & \text{else.} \end{cases} \tag{3.5}$$

Note the similarity with (3.3): Actually, asking whether there exists some compomer of mass M, or if there exists some string of mass M, is equivalent. Computation of table A again requires $O(kM)$ time and $O(M)$ space. In case we do not want to store table A for "future use", memory consumption can be reduced to $O(\max_i a_i)$. Again, our algorithm has pseudo-polynomial running time, linear in M. In contrast, the size of the input is only $\log_2 M$ as this is the number of bits required to encode the number M in memory. Unfortunately, deciding if there is a compomer or a string of mass M is NP-hard [221]: No exact algorithm with running time polynomial in the input size, that is, in k, $\log a_k$, and $\log M$, can exist, unless P = NP. The pseudo-polynomial algorithm introduced above is no contradiction to this hardness result: In fact, the problem is weakly NP-hard, but not strongly [118].

How can we produce a witness, that is, find some compomer c with $\mu(c) = M$? Here and in the following, let $e_i = (0,\dots,0,1,0,\dots,0)$ denote the ith unit vector that has all-zero entries, except for the ith entry, which equals one. Finding a witness is very simple, using table A: Assume that $A[M] = 1$. Start with $c \leftarrow 0$ and $m \leftarrow M$. Find some i such that $m \geq a_i$ and $A[m-a_i] = 1$. Set $c \leftarrow c + e_i$ and $m \leftarrow m - a_i$, and repeat until $m = 0$. Output c. Similarly, we can build a witness string s with $\mu(s) = M$. One can easily see that this algorithm is correct, and has running time $O(k\frac{M}{a_1})$.

Example 3.4. Consider again the weighted alphabet $\Sigma = \{2,3,7,10\}$. We want to compute a witness c for mass $M = 13$. Here is table A, modified from Example 3.3:

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$A[m]$	1	0	1	1	1	1	1	1	1	1	1	1	1	1

We start with $m \leftarrow 13$ and $c = (0,0,0,0)$. For $i = 1$ we find $A[m-a_1] = A[13-2] = 1$, so we set $m \leftarrow m - a_1 = 11$ and $c \leftarrow (1,0,0,0)$. We repeat this four more times and reach $m = 3$ and $c = (5,0,0,0)$. Now, $A[m-a_1] = 0$, but for $i = 2$ we have again $A[m-a_2] = A[3-3] = 1$. We set $m \leftarrow m - a_2 = 0$ and $c \leftarrow (5,0,0,0) + (0,1,0,0) = (5,1,0,0)$, and we are done. As desired, $\mu(c) = 5 \cdot 2 + 1 \cdot 3 = 13$.

```
1: procedure FindAllRec(integer i ≤ k, mass m, compomer c)
2:     if i = 0 then
3:         Output c and return
4:     end if
5:     if B[i − 1, m] = 1 then
6:         FindAllRec(i − 1, m, c)
7:     end if
8:     if m ≥ a_i and B[i, m − a_i] = 1 then
9:         FindAllRec(i, m − a_i, c + e_i)
10:    end if
11: end procedure
```

Algorithm 3.2: Recursive algorithm for enumerating all compomers of a given mass m. To decompose mass M, this algorithm is initially called as FindAllRec$(k, M, 0)$.

3.5 Enumerating strings and compomers

We consider the question most interesting for the majority of MS applications: Given a mass M, find all strings s with $\mu(s) = M$, and find all compomers c with $\mu(c) = M$. First, we consider creating all strings of mass M. This problem can be easily solved by backtracking through the table C': We start with the empty string and mass m, and recursively add characters at the right end. You may also add all characters on the left, if you prefer it this way. Now, we test for each character a_i with $i = 1, \ldots, k$ whether $C'[m - a_i] > 0$ holds, meaning that there is at least one string of this mass. If this holds, we recursively add character a_i to the string and recurse with mass $m - a_i$. We output a string every time $m = 0$ holds. Compare this with Algorithm 3.2 below: There, we just have to consider two cases; for generating strings, we recurse into $|\Sigma|$ many cases. We have formulated this as a recursive algorithm but we can also make it an iterative one, see Exercise 3.9 and below. (This algorithm uses the generated string as a "stack", so it may be up for discussion if the algorithm is truly "iterative".)

Next, we consider the problem of enumerating all compomers c with $\mu(c) = M$. This problem can again be solved by backtracking, this time through the table C: For that, we consider the two "buckets" of recurrence (3.2), and follow both cases to actually compute the individual compomers. As for the decision problem, a binary table is sufficient for this task, as only requests of the form "$C[i, m] > 0$?" have to be answered. Unfortunately, we cannot use table A to enumerate all compomers, as the information stored there is not sufficient. To this end, we define a third, two-dimensional binary table B. We define $B[i, m] = 1$ if and only if there is at least one compomer of mass m over the sub-alphabet $\{a_1, \ldots, a_i\}$. Now, we initialize $B[0, 0] = 1$ and $B[0, m] = 0$ for $m = 1, \ldots, M$, and use the recurrence

$$B[i, m] = \begin{cases} 1 & \text{if } B[i-1, m] = 1 \text{ or } B[i, m - a_i] = 1 \text{ for } m \geq a_i, \\ B[i-1, m] & \text{for } m < a_i. \end{cases} \tag{3.6}$$

We could also write "$\max\{B[i-1, m], B[i, m - a_i]\}$ for $m \geq a_i$" in the first line of recurrence (3.6), representing a logical "OR". Clearly, $B[i, m] = 1$ holds if and only if $C[i, m] > 0$.

See Algorithm 3.2 for the pseudocode of the enumeration algorithm. Clearly, we can replace the statement "$B[i, m] = 1$" by "$C[i, m] > 0$". The algorithm is written as a recursion for the sake of simplicity, but we can also do this task iteratively, see Algorithm 3.5 on page 54. These algorithms can be easily modified to take into account upper and lower bounds for each character, see Exercise 3.21.

How much time is needed to compute all compomers? Comparable to the algorithm for computing a single witness, the algorithm FindAllRec requires $O(k\frac{M}{a_1})$ time *per decomposition*. So, the running time is linear in the size of the output, which is quite obvious: the larger the output, the longer the running time. But even if there are only a few compomers with mass M, the running time also depends linearly on M, which is somewhat unfavorable. In the next section, we will get to know a different approach that does not have this unfavorable property.

3.6 The Money Changing problem and the Round Robin algorithm

In Sec. 3.4, we asked if there is at least one compomer c with mass $\mu(c) = M$. This problem was first posed in 1884 and is known as the MONEY CHANGING PROBLEM. To understand what this has to do with money changing, assume that we live in a country where only coins with values $a_1, \ldots, a_k$ such that $a_1 < a_2 < \cdots < a_k$ are available. We want to know what change can be given with these coins. This problem is trivial if a coin with value $a_1 = 1$ exists. Let $g := \gcd(a_1, \ldots, a_k)$ be the *greatest common divisor* of numbers $a_1, \ldots, a_k$: So, g is a divisor of each a_i for all $i = 1, \ldots, k$, and g is the largest such integer. In case $g > 1$ then it is easy to see that we can only make change for numbers $0, 1g, 2g, 3g, \ldots$. In the following, we will usually assume $\gcd(a_1, \ldots, a_k) = 1$. It turns out that the results of this section can also be applied in case $\gcd(a_1, \ldots, a_k) > 1$, see the end of the section for the simple details.[4]

Let $\Sigma = \{a_1, \ldots, a_k\}$ be a fixed weighted alphabet with $\gcd(a_1, \ldots, a_k) = 1$. We want to decide whether some mass M is decomposable or not over Σ. For an integer M, we write $r = M \mod a_1$ for the *residue* of M modulo a_1: This is the unique number $r \in \{0, \ldots, a_1 - 1\}$ such that $M = qa_1 + r$ for some integer q. For $M \geq 0$ we easily see that $q = \lfloor M/a_1 \rfloor$. We say that M belongs to *residue class* r (modulo a_1).

A simple observation is as follows: If M is decomposable, then $M + a_1, M + 2a_1, M + 3a_1, \ldots$ are also decomposable. (It holds that $M + a_i$ is decomposable for any $i = 1, \ldots, k$, but we only need the statement for $i = 1$.) This implies that there is a smallest such mass that is decomposable. For each residue class $r = 0, \ldots, a_1 - 1$, let $N[r]$ be the smallest mass that is decomposable satisfying $N[r] \mod a_1 = r$. So, $N[0 \ldots a_1 - 1]$ is a one-dimensional array satisfying

$$N[r] = \min\{n : r = n \mod a_1, \text{ and } n \text{ is decomposable over } \{a_1, \ldots, a_k\}\}$$

for $r = 0, \ldots, a_1 - 1$. Here, $N[r] = +\infty$ if no such number exists, and the minimum is empty. Clearly, $\mu(c) = N[r]$ for some compomer $c = (c_1, \ldots, c_k)$ implies $c_1 = 0$ because otherwise, $N[r] - a_1$ has a decomposition, too. The table $N[0 \ldots a_1 - 1]$ forms the *residue table* of the instance.

Example 3.5. For the remainder of this section, we will consider the weighted alphabet $\Sigma = \{5, 8, 9, 12\}$. The residue table $N[0 \ldots a_1 - 1]$ of this alphabet is:

r	0	1	2	3	4
$N[r]$	0	16	12	8	9

It is straightforward to check that this truly is the residue table of the above instance: Clearly, mass 0 belongs to residue class $r = 0$ (modulo $a_1 = 5$) and obviously, there is no smaller non-negative integer. Next, 16 belongs to residue class $r = 1$ and can be decomposed as $16 = 8 + 8$, whereas we cannot decompose $16 - 5 = 11$. We can continue in this fashion up to residue class $r = 4$, where $9 = 9$ can be decomposed but $9 - 5 = 4$ cannot.

Assume that we know the residue table $N[0 \ldots a_1 - 1]$ of a weighted alphabet: This allows us to answer the question "is mass M decomposable?" in *constant* time. We simply calculate $r \leftarrow M \mod a_1$; then, M is decomposable if and only if $M \geq N[r]$. For example, assume that we want to know if 17 can be decomposed over the weighted alphabet from Example 3.5. We calculate $17 \mod 5 = 2$ and check $17 \geq N[2] = 12$, so the answer is "yes". On the other hand, we cannot decompose 11, as $11 \mod 5 = 1$, and $11 < N[1] = 16$.

Before we concentrate on computing the residue table, we take a short detour: Recall that $\gcd(a_1, \ldots, a_k) = 1$. Then there exists a number $g := g(a_1, \ldots, a_k)$, called the *Frobenius number*, such that g cannot be decomposed, but *all* masses $M > g$ can be decomposed. It might be somewhat surprising that for an arbitrary weighted alphabet, all sufficiently large masses can be decomposed. See Exercise 3.15 for the proof of the special case $k = 2$. Given the residue table $N[0 \ldots a_1 - 1]$ of an instance, there is a simple

[4] Many countries got rid of small coins, but since supermarkets love prices such as \$0.99, the amount you have to pay has to be rounded.

r	$a_1 = 5$	$a_2 = 8$	$a_3 = 9$	$a_4 = 12$
0	0	0	0	0
1	∞	16	16	16
2	∞	32	17	12
3	∞	8	8	8
4	∞	24	9	9

Figure 3.1: Extended residue table $N[0\ldots4, 0\ldots4]$ of the weighted alphabet $\Sigma = \{5, 8, 9, 12\}$ from Example 3.5, as well as iterations of the Round Robin algorithm.

formula to compute the Frobenius number g, as well as the number ω of *omitted* values that cannot be decomposed over $\Sigma = \{a_1, \ldots, a_k\}$:

$$g = \max_{r=0,\ldots,a_1-1} \{N[r]\} - a_1 \quad \text{and} \quad \omega = \sum_{r=0}^{a_1-1} \left\lfloor \frac{N[r]}{a_1} \right\rfloor = \frac{1}{a_1} \sum_{r=0}^{a_1-1} N[r] - \frac{a_1 - 1}{2}. \tag{3.7}$$

For the second equation, remember that $N[r]$ belongs to residue class r (modulo a_1) and, hence,

$$\sum_{r=0}^{a_1-1} \left\lfloor \frac{N[r]}{a_1} \right\rfloor = \sum_{r=0}^{a_1-1} \frac{N[r] - r}{a_1} = \sum_{r=0}^{a_1-1} \frac{N[r]}{a_1} - \sum_{r=0}^{a_1-1} \frac{r}{a_1} = \frac{1}{a_1} \sum_{r=0}^{a_1-1} N[r] - \frac{1}{a_1} \cdot \frac{(a_1-1)(a_1-1+1)}{2}.$$

Equations (3.7) may appear somewhat surprising but in fact, it is straightforward to show that these identities hold; see Exercise 3.16. For the weighted alphabet from Example 3.5, we can read from the residue table that $g = 16 - 5 = 11$, and there are

$$\omega = \frac{16 + 12 + 8 + 9}{5} - \frac{4}{2} = 9 - 2 = 7$$

masses without a decomposition; namely, these are masses 1, 2, 3, 4, 6, 7, 11.

Now, the interesting question is: Given a weighted alphabet $\{a_1, \ldots, a_k\}$, how can we efficiently calculate its residue table $N[0\ldots a_1-1]$? This can be achieved by the *Round Robin* algorithm: We compute the values of N iteratively for the sub-problems "Find $N[0\ldots a_1 - 1]$ for the instance $\{a_1, \ldots, a_i\}$", for $i = 1, \ldots, k$. For $i = 1$ we can only decompose masses of the form $M = qa_1$ whereas all other masses cannot be decomposed. Hence, we start with $N[0] = 0$ and $N[r] = \infty$ for $r = 1, \ldots, a_1 - 1$. When constructing the residue table for the next step, the current values $N[0\ldots a_1-1]$ are updated. Suppose we know the correct values $N'[r]$ for the sub-problem $\{a_1, \ldots, a_{k-1}\}$, and we want to calculate those of the original problem $\{a_1, \ldots, a_k\}$. We first concentrate on the simple case that $\gcd(a_1, a_k) = 1$. We initialize $N[r] \leftarrow N'[r]$ for all $r = 0, \ldots, a_1 - 1$, and $n \leftarrow N[0] = 0$. In every step of the algorithm, set $n \leftarrow n + a_k$ and $r \leftarrow n \bmod a_1$. Let $n \leftarrow \min\{n, N[r]\}$ and $N[r] \leftarrow n$. We repeat this loop until n equals 0. In case all $a_2, \ldots, a_k$ are coprime to a_1 — that is, $\gcd(a_1, a_i) = 1$ holds for all $i = 2, \ldots, k$ — then this short algorithm is already sufficient to find the correct values $N[r]$.

Example 3.6. Consider the weighted alphabet $\Sigma = \{5, 8, 9, 12\}$ from Example 3.5. In Figure 3.1, each column can be viewed as representing one iteration of the Round Robin algorithm. For example, focus on the column $a_3 = 9$. We start with $n = 0$. In the first step, we have $n \leftarrow 9$ and $r = 4$. Since $n < N[4] = 24$ we update $N[4] \leftarrow 9$. Second, we have $n \leftarrow 9 + 9 = 18$ and $r = 3$. In view of $n > N[3] = 8$ we set $n \leftarrow 8$. Third, we have $n \leftarrow 8 + 9 = 17$ and $r = 2$. Since $n < N[2] = 32$ we update $N[2] \leftarrow 17$. Fourth, we have $n \leftarrow 17 + 9 = 26$ and $r = 1$. In view of $n > N[1] = 16$ we set $n \leftarrow 16$. Finally, we return to $r = 0$ via $n \leftarrow 16 + 9 = 25$.

It is straightforward how to generalize the algorithm for $d := \gcd(a_1, a_i) > 1$: In this case, we do the updating independently for every residue $p = 0, \ldots, d - 1$: Only those $N[r]$ for $r \in \{0, \ldots, a_1 - 1\}$ are

```
1:  procedure RoundRobin(weighted alphabet Σ)
2:      initialize N[0] ← 0 and N[r] ← ∞ for r = 1, ..., a_1 − 1
3:      for i ← 2, ..., k do
4:          d ← gcd(a_1, a_i)
5:          for p ← 0, ..., d − 1 do
6:              find n = min{N[q] : p = q mod d, 0 ≤ q ≤ a_1 − 1}
7:              if n < ∞ then
8:                  for j ← 1, ..., a_1/d − 1 do          ▷ repeat a_1/d − 1 times
9:                      n ← n + a_i
10:                     r = n mod a_1
11:                     n ← min{n, N[r]}
12:                     N[r] ← n
13:                 end for
14:             end if
15:         end for
16:     end for
17: end procedure
```

Algorithm 3.3: Constructing the residue table $N[0 \ldots a_1 - 1]$ of a weighted alphabet $\Sigma = \{a_1, \ldots, a_k\}$.

updated that satisfy $r = p \mod d$. To guarantee that the Round Robin loop completes updating after a_1/d steps, we have to start the loop from a minimal $N[r]$ with $r = p \mod d$. For $p = 0$ we know that $N[0] = 0$ is the unique minimum, while for $p \neq 0$ we search for the minimum first. See Algorithm 3.3 for the pseudocode of the algorithm. The inner loop (lines 8–13) will be executed only if the minimum $\min\{N[q]\}$ is finite; otherwise, the elements of the residue class cannot be decomposed over $a_1, \ldots, a_i$ because of $\gcd(a_1, \ldots, a_i) > 1$.

It is quite easy to see that the Round Robin algorithm computes the residue table of a weighted alphabet $\Sigma = \{a_1, \ldots, a_k\}$ in $\Theta(k\, a_1)$ time; besides $O(a_1)$ memory for storing the current residue table, we need only constant extra memory.

Example 3.7. Consider the weighted alphabet $\Sigma = \{6, 7, 8\}$. Now, $\gcd(a_1, a_3) = 2$ so for $i = 3$, we have to compute the residue table $N[0 \ldots a_1 - 1]$ in two independent round robin runs. The residue tables computed by the iterations of the Round Robin algorithm are:

r	$a_1 = 6$	$a_2 = 7$	$a_3 = 8$
0	0	0	0
1	∞	7	7
2	∞	14	8
3	∞	21	15
4	∞	28	16
5	∞	35	23

For the last column, we first consider $p = 0$: Then, $n = \min\{0, 14, 28\} = 0$. We repeat two times: In the first step, we set $n \leftarrow 0 + 8 = 8$ and $r = 2$. Since $n < N[2] = 14$ we update $N[2] \leftarrow 8$. Second, we have $n \leftarrow 8 + 8 = 16$ and $r = 4$. In view of $n < N[4] = 28$ we set $N[4] \leftarrow 16$. The next step would bring us back to $r = 0$. Next, we consider $p = 1$: Here, $n = \min\{7, 21, 35\} = 7$. We again repeat two times: In the first step, we set $n \leftarrow 7 + 8 = 15$ and $r = 3$. Since $n < N[3] = 21$ we update $N[3] \leftarrow 15$. Second, we have $n \leftarrow 15 + 8 = 23$ and $r = 5$. In view of $n < N[5] = 35$ we set $N[5] \leftarrow 23$. The next step would bring us back to $r = 1$, and we are done.

For enumerating *all* decompositions of mass M, the information contained in the residue table is unfortunately insufficient. But to our delight, we have implicitly come up with a data structure that allows

```
1:  procedure FindAllERT(integer i ≤ k, mass m, compomer c)
2:      if i = 1 then
3:          c_1 ← c_1 + m/a_1; Output c and return
4:      end if
5:      r ← m mod a_1
6:      if m ≥ N[i − 1, r] then
7:          FindAllERT(i − 1, m, c)
8:      end if
9:      r ← (m − a_i) mod a_1
10:     if m ≥ a_i and m − a_i ≥ N[i, r] then
11:         FindAllERT(i, m − a_i, c + e_i)
12:     end if
13: end procedure
```

Algorithm 3.4: Recursive algorithm for enumerating all compomers of a given mass m, based on the extended residue table $N[0 \ldots k, 0 \ldots a_1 - 1]$. To decompose mass M, this algorithm is initially called as FindAllERT$(k, M, 0)$.

us to tackle the enumeration problem: Namely, for each $r = 0, \ldots, a_1 - 1$ and each $i = 1, \ldots, k$, we search for the smallest number $N[i, r]$ such that $r = N[i, r] \mod a_1$, and $N[i, r]$ is decomposable over $\{a_1, \ldots, a_i\}$. Formally, we define the *extended residue table* $N[0 \ldots k, 0 \ldots a_1 - 1]$ to be a two-dimensional table such that

$$N[i, r] = \min\{n : r = n \mod a_1, \text{ and } n \text{ is decomposable over } \{a_1, \ldots, a_i\}\}$$

where $N[i, r] = +\infty$ if no such number exists, and the minimum is empty. Clearly, space for storing the extended residue table is $O(ka_1)$. Here, the nice feature is that there is no "largest mass" that we have to decide upon during preprocessing.

The nice feature of the Round Robin algorithm is that we have already computed the extended residue table of the instance: We simply have to store each iteration of the algorithm as a "column" of the matrix, when iterating $i = 0, \ldots, k$. See Fig. 3.1 for the extended residue table of Example 3.5.

We can easily use the extended residue table to enumerate all decompositions: In fact, we simply have to replace the query "$B[i, m] = 1$" in Algorithm 3.2 by the equivalent query "$m \geq N[i, r]$ for $r = m \mod a_1$". The recursion stops at $i = 1$, where m must be a multiple of a_1. See Algorithm 3.4 for the result. In fact, we easily transform the iterative variant of that algorithm, namely Algorithm 3.5, into an iterative variant using the extended residue table, see Exercise 3.11.

So, we have improved upon the memory consumption of our approach, as well as the running time during preprocessing. Also, the new algorithm has the desirable property that we do not have to decide upon a largest mass that we want to decompose during preprocessing. Instead, for a fixed weighted alphabet, we compute its unique extended residue table; this allows us to compute decompositions for *any* mass m at a later stage. It turns out that the resulting algorithm is also about twice as fast in practice [92], at least for certain applications: This is due to the reduced memory consumption, which allows us to store the extended residue table in the processor cache, instead of having to store array B in main memory, see Sec. 8.1.

One thing that we have not improved upon, is the running time per decomposition. But there is a modification of the algorithm so that we can guarantee that every decomposition is computed in $O(ka_1)$ time: To this end, we do not recurse in an arbitrary order but instead, treat all recursions for each residue class in one batch. Whereas the resulting algorithm allows us to prove an improved worst-case running time, the overhead required for processing the residue classes individually, is usually too high in applications. The algorithm can be found in Fig. 4 of [34], we defer further details.

Finally, a few words about the case $g := \gcd(a_1, \ldots, a_k) > 1$ that we have ignored so far. To cover this case is rather simple: Replace masses $a_1, \ldots, a_k$ by new masses $a_1/g, \ldots, a_k/g$, and construct the

(extended) residue table for this weighted alphabet. If you want to decompose a mass M (or decide if it is decomposable), first check if $M \mod g = 0$ holds: Otherwise, M has no decomposition over $a_1, \dots, a_k$. Next, decompose the mass M/g over the alphabet $a_1/g, \dots, a_k/g$; all decompositions that you compute, are also decompositions of M over $a_1, \dots, a_k$.

3.7 Approximating the number of compomers

Before we start this section, a word of warning is in order. The term "f approximates g" for two functions $f, g : \mathbb{N} \to \mathbb{R}$ can be used with many different meanings: In computer science, this means that we can calculate some number with a *guaranteed* relative error, so $f(n) \leq (1+\varepsilon)g(n)$ or $f(n) \geq (1-\varepsilon)g(n)$ for *all* $n \in \mathbb{N}$. Here, $\varepsilon > 0$ can be a constant or a function depending on n. In mathematics, this sometimes means that f and g are *asymptotically equivalent* or *asymptotically equal*, so $\lim_{n\to\infty} f(n)/g(n) = 1$, which is denoted $f \sim g$. Hence, f "behaves like" g and as n goes towards infinity, the relative error goes to zero. Be aware that in both cases, we only consider relative errors: The absolute difference may be huge and will often go to infinity as $n \to \infty$, see Exercise 3.24. Be also warned that $f \sim g$ does not tell us how fast the error goes to zero, and that the approximation might be arbitrarily bad for the first N numbers where, again, N might be arbitrarily large. See Sec. 15.7 for more details.

Let $\gamma(M)$ denote the number of compomers with mass exactly M, over some fixed alphabet Σ. Often, we do not have to compute $\gamma(M)$ exactly but rather, want to compute a reasonable estimate. Luckily, there is a simple formula that can help us to estimate the number of decompositions in constant time. The following result is due to Issai Schur:

Theorem 1. *If* $\gcd(a_1, \dots, a_k) = 1$ *then*

$$\gamma(M) \sim \frac{1}{(k-1)!a_1 a_2 \cdots a_k} M^{k-1} \quad \text{for } M \to \infty. \tag{3.8}$$

Actually, we can infer from this theorem that every sufficiently large number M is decomposable over Σ. Unfortunately, convergence is rather slow. A better approximation was given by Beck, Gessel and Komatsu [15]:

$$\gamma(M) \approx b_{k-1}M^{k-1} + b_{k-2}M^{k-2} + b_{k-3}M^{k-3} \dots \tag{3.9}$$

where

$$\begin{aligned} b_{k-1} &:= \frac{1}{a_1 \cdots a_k} \cdot \frac{1}{(k-1)!} \\ b_{k-2} &:= \frac{1}{a_1 \cdots a_k} \cdot \frac{1}{2(k-2)!} \cdot \sum_{i=1}^{k} a_i \\ b_{k-3} &:= \frac{1}{a_1 \cdots a_k} \cdot \frac{1}{4(k-3)!} \cdot \left(\frac{1}{3} \sum_{i=1}^{k} a_i^2 + \sum_{i<j} a_i a_j \right) \end{aligned} \tag{3.10}$$

In fact, the authors show how to compute all the coefficients of the polynomial, we omit the details. See Sec. 8.5 on how this can be used to estimate the number of molecular formulas over some alphabet of elements.

We mentioned that the above estimates do not give any guarantees, such as: The approximation will in all cases be at most twice as large as the number of decompositions. Theorem 1 guarantees that the relative error will drop to zero only when M goes to infinity. As we will see in Sec. 8.6, Eq. (3.9) cannot be used to give a reasonable estimate of the number of amino acid decompositions. By contrast, even the simpler approximation (3.8) gives the right order of magnitude for the number of molecular formulas, and (3.9) results in reliable estimates (Sec. 8.5), if we ignore the fluctuation of this number due to the combinatorial nature of the problem.

We will now turn to approximations that give us a guarantee on how large the relative error is. Let $\Gamma(M)$ be the number of decompositions with mass $m \leq M$, so $\Gamma(M) = \sum_{m=0}^{M} \gamma(m)$. Dyer [94] gave a *Fully*

Polynomial Randomized Approximation Scheme (FPRAS) for this number: We choose an arbitrary relative error $\varepsilon > 0$, then the algorithm computes an estimate $\bar{\Gamma}(M)$ such that, with high probability,

$$(1-\varepsilon)\Gamma(M) \le \bar{\Gamma}(M) \le (1+\varepsilon)\Gamma(M)$$

in time $O(k^5 + \varepsilon^{-2}k^4)$. So, we can approximate $\Gamma(M)$ with arbitrary precision, where we trade running time for precision. Note that M itself is no longer part of the running time. Unfortunately, this will not lead to an approximation for the number of compomers with mass *exactly* M. In fact, one can easily see that no PTAS can exist for this number: The reason is that if we can approximate $\gamma(M)$ with performance ratio $\varepsilon = \frac{1}{2}$ in polynomial time, then we can decide in polynomial time whether $\gamma(M) = 0$ holds. We noted above that this is not possible unless P = NP.

3.8 Historical notes and further reading

Our presentation in this chapter roughly follows the paper of Böcker and Lipták [34], see there for additional details and missing proofs.

The idea of using compomers for the analysis of mass spectrometry data, dates back at least to the 1980s: Back in 1984, Sakurai *et al.* [305] used compomers over the amino acid alphabet for *de novo* sequencing of peptides. In their approach, they took an MS/MS spectrum of an unknown peptide with precursor mass M, generated all compomers c with $\mu(c) = M$, then generated all strings s with $\text{comp}(s) = c$ and, finally, simulated a reference spectrum for each such string s to compare it against the measured spectrum. Obviously, this approach suffered heavily from the huge number of compomers over an alphabet with 19 characters.

The Money Changing problem and, in particular, the problem of computing Frobenius numbers has been around in mathematics for quite some time: In 1884, Sylvester asked for the Frobenius number of $k = 2$ coins a_1, a_2, and Curran Sharp showed that $g(a_1, a_2) = a_1a_2 - a_1 - a_2$ [340]. For three coins a_1, a_2, a_3, Greenberg [135] and Davison [73] independently discovered simple algorithms with fast running times. Kannan [172] established algorithms that for *any fixed* k, compute the Frobenius number in time polynomial in $\log a_k$. Unfortunately, the running time has a double exponential dependency on k, and is not competitive for $k \ge 5$. Reading the Frobenius number from the residue table was suggested by Brauer and Shockley [40]. The Round Robin algorithm and the extended residue table were introduced by Böcker and Lipták [32, 33].

In 2007, Einstein, Lichtblau, Strzebonski and Wagon [100] presented an elaborate method that can solve instances with $k = 4$ and $a_k \le 10^{100}$ in under one second, and remains efficient up to $k = 8$ for numbers with 100 digits. Other methods (in particular, computing the residue table) can be faster if k is large whereas a_1 is relatively small [16]. Computing the Frobenius number is NP-hard [279], so we cannot hope to find algorithms polynomial in k and $\log a_k$ simultaneously unless P = NP. Many results regarding the Money Changing problem and Frobenius numbers are based on generating functions, see [367] for an introduction. There has been considerable work on bounds for Frobenius numbers, see Ramírez-Alfonsín [280] for a survey.

The solution of the Change Making problem (see Exercise 3.1) was proposed by Gilmore and Gomory [126] in 1965, but it is probably easier to come up with a solution yourself than to find it in their paper.

The Money Changing problem is also closely related to unbounded integer knapsacks [231]: There, one replaces the condition $\sum_j c_j a_j = M$ by $\sum_j c_j a_j \le M$. In fact, the approximation result of Dyer [94] mentioned in Sec. 3.7 is for unbounded integer knapsacks. Although these problems look similar, algorithms for solving unbounded integer knapsacks such as the algorithm of Martello and Toth [230], cannot be used for the Money Changing problem.

Algorithm 3.5 is the iterative version of the FindAll algorithm. Be aware that, although it is more complicated than the recursive Algorithm 3.2, it is faster in practice, as we avoid the stack handling; but Dührkop *et al.* [92] found the speedup to be merely 1.4-fold, with both algorithms using the extended residue table. The fastest variant is hard-coding $|\Sigma|$ many While-loops. In practice, both approaches will

```
1:  procedure FindAllIt(mass m)
2:      compomer c = (c_1, ..., c_k) ← 0
3:      integer i ← k
4:      while i ≤ k do                                  ▷ recall B[i, m] = 0 for m < 0
5:          if B[i, m] = 0 then                         ▷ is this decomposable at all?
6:              while i ≤ k and B[i, m] = 0 do          ▷ no, go to next one
7:                  m ← m + c_i a_i
8:                  c_i ← 0
9:                  i ← i + 1
10:             end while                               ▷ now, B[i, m] = 1 holds
11:             if i ≤ k then
12:                 m ← m − a_i
13:                 c_i ← c_i + 1
14:             end if
15:         else                                        ▷ yes, decomposable
16:             while i > 1 and B[i − 1, m] = 1 do      ▷ initially, we do not add any coins
17:                 i ← i − 1
18:             end while                               ▷ now, B[i, m] = 1 but B[i − 1, m] = 0
19:             if i = 1 then
20:                 c_1 ← m/a_1
21:                 Output c = (c_1, ..., c_k)
22:                 i ← 2
23:             end if
24:             if i ≤ k then                           ▷ move to next decomposition
25:                 m ← m − a_i
26:                 c_i ← c_i + 1
27:             end if
28:         end if
29:     end while
30: end procedure
```

Algorithm 3.5: Iterative algorithm for enumerating all compomers of a given mass m. To decompose mass M, this algorithm is initially called as FindAllIt(M). We assume $B[i, m] = 0$ for all $m < 0$.

show a comparable running time, as compiler optimizations such as loop unrolling cannot be performed here, see Exercise 3.20.

3.9 Exercises

3.1 Assume you are given an infinite supply of coins with values $\Sigma = \{2, 3, 7, 10\}$ dollars. How can you make change for 18 dollars with as few coins as possible? Provide a general solution to the problem. This problem is known in computer science as the Change Making problem, and can be solved with a recurrence similar to (3.2).

3.2 Compute the residue table and the Frobenius number for the weighted alphabet $\Sigma = \{6, 9, 20\}$. How can you "make change" for 41 "dollars"? This particular problem is also known as Chicken McNuggets problem — explain why, and discuss it with your benchmate.

3.3 You can compute Frobenius numbers using the search engine Wolfram Alpha at http://www.wolframalpha.com/. What is the Frobenius number of the alphabet {12312312, 4567456745, 678678678, 4567894567}?

3.4 Show by examples that the greedy algorithm cannot optimally solve the MONEY CHANGING and the CHANGE MAKING problem.

3.5 How many strings can be made using all characters of the string ALGORITHMUS exactly once? As an example, there are three strings for the input string ABA, namely AAB, ABA, and BAA. How many strings can be made from ABRACADABRA? Try to find a formula for this number.

3.6 Let Σ be a weighted alphabet with integer masses $\mu : \Sigma \to \mathbb{N}_{>0}$, where not all masses are necessarily different. Build an algorithm that decomposes some mass M over this alphabet, using any of the FINDALL algorithms as a subroutine.

3.7 Let $\Sigma = \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\}$ be a weighted alphabet with masses $\mu(\mathsf{a}) = 3$, $\mu(\mathsf{b}) = 6$, $\mu(\mathsf{c}) = 8$, and $\mu(\mathsf{d}) = 9$. Compute all compomers of mass $M = 23$ using the recursive algorithm FINDALLREC (Algorithm 3.2). List all calls of the algorithm, in the order in which they are executed. Here, we use the "old fashioned" version of weighted alphabets, to make it easier to write up the compomers.

3.8 Let Σ be the weighted alphabet from the previous exercise. Compute all compomers of mass $M = 23$ using the iterative algorithm FINDALLIT (Algorithm 3.5). List values of variables i, m, and c for each entry into the WHILE-loop (line 5), in the order in which the algorithm is executed.

★ 3.9 Come up with an "iterative" variant of the algorithm for finding all strings of a given mass M using table $C'[m]$, similar to Algorithm 3.5.

3.10 Let $\Sigma := \{6, 7, 17, 22\}$ be a weighted alphabet. Compute the ERT table using the Round Robin algorithm, and use the ERT table to compute all decompositions of mass 35. Compute the Frobenius number and the number of omitted values of this instance.

3.11 Modify Algorithm 3.5 so that it uses the extended residue table instead of array B, similar to Algorithms 3.2 and 3.4.

3.12 Assume that we have computed $C'[m]$ for all $m = 0, \ldots, M$ as the number of strings over an alphabet Σ. How many strings of precursor mass M have a prefix of mass m, and how many have a suffix of mass m? Finally, how many strings of precursor mass M have a prefix or suffix (or both) of mass $m \leq M/2$? Hint: The solution to all three questions is very simple and, in particular, you do not need a new recurrence.

3.13 Using integer masses, find the prefix and suffix of a peptide with smallest mass so that, with the "true" *de novo* sequencing mass modification ±0 and +18, both have identical mass, violating Assumption 3 from Chapter 2. Argue why the string resulting from appending prefix and suffix, is truly the string of smallest mass violating the assumption.

★ 3.14 Using integer masses, count the number of peptide strings of mass M that have a prefix of mass m and a suffix of mass $m+18$, for any $m \in \{0, \ldots, M\}$. If you have previously computed $C'[0 \ldots M]$, you can do so in $O(M^2)$ time. To come up with useful numbers, you should treat characters I and L as one; similarly, characters K and Q. Plot the relative number of such strings against M, for $M = 0, \ldots, 3500$.

★ 3.15 Prove that for a weighted alphabet $\{a_1, a_2\}$ that the number $g = g(a_1, a_2) = a_1 a_2 - a_1 - a_2$ cannot be decomposed, but *all* $M > g$ can be decomposed.

3.16 Prove the correctness of Eq. (3.7).

3.17 Write a program to compute arrays B and C for the amino acid alphabet. As integer weights, use those from Table 2.1 times 100, rounded to the closest integer. Compute $B[0 \ldots 20000]$ and $C[0 \ldots 20000]$.

★ 3.18 Compute the residue table for the amino acid alphabet in Table 2.1, rounded to integers. What is the Frobenius number, that is, the largest mass that does not have a decomposition?

3.19 The previous exercise is not complicated, but it is a lot of work. Write and run a computer program that does the job for you. Rerun your program but this time, multiply all masses by 1000 before rounding.

3.20 Implement the recursive and iterative algorithms for enumerating compomers, Algorithms 3.2 and 3.5, as well as 19 nested WHILE-loops for the amino acid alphabet. Using each algorithm, enumerate all compomers for integer masses $m = 0, \ldots, 20000$, using the array B from Exercise 3.17. Compare running times. Warning: Do not print out compomers, as this will by far exceed the time required to compute them.

3.21 It is easy to modify all presented algorithms for enumerating compomers, when upper and lower bounds for each character in Σ are given. Show how this can be done. Note that for lower bounds, you do not need *any* changes to the actual algorithms.

3.22 Assume that Σ is an alphabet of integer masses, such that at least one character has positive mass, and at least one character has negative mass. Prove: If some mass M has at least one decomposition, then it has an infinite number of decompositions.

3.23 Assume that Σ is an alphabet of integer masses, positive or negative. Find a recurrence that computes, for any integer mass $M \in \{-M', \ldots, 0, \ldots, M\}$, the minimum number of characters needed to reach this mass.

3.24 Let $f(n) := 2^n + n + 1000$ and $g(n) := 2^n$. Show that $f \sim g$. Compute the absolute and relative error for $n = 1, \ldots, 20$.

4 Comparing Mass Spectra and Database Searching

Claude helped!

> *"If it looks like a duck, and quacks like a duck, we have at least to consider the possibility that we have a small aquatic bird of the family Anatidae on our hands."* (Douglas Adams, *Dirk Gently's Holistic Detective Agency*)

Given a measured sample spectrum and a database with reference spectra, there are two questions to be answered: Which reference spectrum matches best with the measured spectrum? And, how certain are we that this identification is correct? In this chapter, we will focus on the first question; the second question will be addressed in Chapters 6 and 5.

In the vast majority of cases, database searching is based on the comparison of two spectra, where one is the spectrum from the database (the *reference*) whereas the other is our measured spectrum (the *query*). We use this comparison to compute a *score* that measures the similarity or dissimilarity between the two spectra. Candidate spectra from the database are then sorted (ranked) by score; if everything goes according to plan, then the highest-scoring candidate is also the correct solution. With the exception of Sec. 4.11, we will not put particular focus on the application of database searching, but rather consider the general task of comparing two spectra.

Besides database searching, other computational questions such as *de novo* sequencing (Chapters 2 and 11) or annotating mass spectra (Chapter 9) also have to compare mass spectra; the methods for comparing spectra presented in this chapter apply to all of them. Also recall that the reference database may be computed "on the fly" (Sec. 2.8). In most cases, it is also not relevant if the biomolecules we are considering are peptides, metabolites, glycans, or something different. For the sake of readability, we will nevertheless often focus on peptides and searching in a peptide database throughout this chapter; doing so will help us to fill our theoretical concepts with some "meat".

We can make the following differentiation with regard to our task: The first differentiation concerns the mass accuracy of our measurements.

- At least one of the two spectra is measured at unit mass accuracy.
- Both the query and the reference spectrum have high mass accuracy.

Clearly, we can only make use of high mass accuracy if both spectra have this property. Next, we have to differentiate based on the availability of peak intensities.

- The reference spectrum has peak intensities.
- The reference spectrum is a "barcode spectrum" without peak intensities.

The query spectrum is an experimental spectrum and contains information about peak intensities; but the reference spectrum can be either an experimental spectrum — in which case we speak of a *spectral library* to search in — or a simulated one, see for example Sec. 2.6. In the latter case, it is often highly nontrivial how to assign peak intensities. Throughout this chapter, we assume that the peak masses of each spectrum are sorted in increasing order.

The third differentiation is rather a classification of the presented scorings: Namely, we can match peaks *one-to-one*, *one-to-many*, or *many-to-many*. This classification only makes sense when both spectra

have high mass accuracy; for unit mass accuracy, all scorings are one-to-one. For *one-to-one* matchings, each peak in the query may be matched against at most one peak in the reference, and vice versa. For *one-to-many* matchings, one peak in the query may be matched against multiple peaks in the reference, but each peak in the reference is matched against at most one peak in the query. This only makes sense if there is a fundamental difference between the two spectra: That is, the query spectrum is a measured spectrum and the reference spectrum is a simulated spectrum. Otherwise, our comparison should be symmetric. Finally, *many-to-many* matchings allow us to match each peak in the query with each peak in the reference. In principle, there is also the possibility to match one or more peaks in the query against at most one peak in the reference, but such many-to-one matchings are rarely of use in practice.

4.1 Unit mass accuracy: The dot product score

The following is for unit mass accuracy. As such, it is a one-to-one scoring. It is most useful when the reference spectrum has peak intensities, but can also be applied for barcode reference spectra.

The conceptually simplest comparison of two spectra is when both spectra have unit mass accuracy. Here, we can think of the two spectra as vectors $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$ of the same length, where missing peaks have intensity zero. We use the *dot product* (also *inner product* and, rarely, *scalar product*) between the two vectors,

$$x^T y = \sum_{m=1}^{n} x_m y_m$$

as our measure of similarity. Here, "x^T" is the transposed vector x, see any book on linear algebra for the details. To guarantee that values are between 0 and 1, we use the norm $\|x\| = \sqrt{x^T x}$ and define the score as

$$\frac{x^T y}{\|x\| \cdot \|y\|}. \tag{4.1}$$

This has also been referred to as the "cosine score" because in Euclidean geometry, this is the cosine of the angle between the two vectors. But hardly anybody transforms it to angles, which would result in values between 0 and $\pi/2$ (or 0° and 90° if you prefer school math). We can get around the normalizing factor in (4.1) if we assume that both vectors have been normalized individually as $x \leftarrow 1/\|x\| \cdot x$ and analogously for y; now, $\|x\| = \|y\| = 1$ and we can use the dot product $x^T y$ without normalization. Depending on what you are doing, normalizing spectra upfront may or may not be a good idea, see below.

It must be understood that the dot product is not the only mathematically sensible way to measure the (dis)similarity between two vectors; the Euclidean distance $\|x - y\|$ is another well-known possibility. But dot products work well for comparing (mass) spectra and do not require a geometric interpretation (Euclidean space).

In practice, many scientists have tried to improve on the dot product for searching in spectral libraries. Two modifications that were suggested repeatedly are a "weighting factor" ψ and an "intensity transformation" φ. Weighting factor $\psi : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ allows us to put different weights $\psi(m)$ (assign importance) to different masses m of the spectrum. The intensity transformation $\varphi : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a monotonic function that is most often used to "tone down" the effect of intensities and intensity differences: Usual choices are $\varphi(x) = \sqrt{x}$ or, more generally, $\varphi(x) = x^c$ for $c \in (0, 1)$, but $\varphi(x) = \max\{0, \log(x) + d\}$ for some $d > 0$ is also a reasonable choice. Now, the dot product becomes

$$k(x, y) := \sum_{m=1}^{n} \psi(m)\, \varphi(x_m)\, \varphi(y_m) \tag{4.2}$$

and the cosine score (4.1) has to be modified accordingly. Similar modifications can be applied for other scores presented below. For the sake of readability, we will ignore weighting factors ψ in our presentation, and we assume that intensity transformations $x \leftarrow \varphi(x)$ *have been applied to the spectra.* (In your code, you should do it *exactly the other way round*: Integrate the intensity transformation into the function that

computes the dot product etc., but *never* apply it to the spectra. Do not photoshop your spectra, unless you know exactly what you are doing. Say, deep algorithm engineering.)

Arguably the one thing to remember about these dot product modifications is reweighting intensities: Replacing intensities by their square roots has found extremely wide application in the community. In fact, if you read in a paper that spectra were compared using the dot product, there is a 50 % chance that square root intensities were used. One should always keep that in mind. Also, unless you have a good reason not to do so, *you should also use square root intensities when you are calculating dot products* between spectra.

4.2 On the eternal beauty of dot products

Using dot products for spectrum comparison, has a long history in computational mass spectrometry. This extends far beyond database search, and well into the realm of machine learning. Why is that?

It turns out that dot products (or, more precisely speaking, inner products) have certain properties which are highly desirable both from the theoretical (mathematical) and from the practical standpoint. An *inner product* $k : V \times V \to \mathbb{R}$ over the set of real numbers $\mathbb{R}$ and vector space V satisfies three conditions:

- symmetry, $k(x, y) = k(y, x)$
- linearity, $k(ax + by, z) = ak(x, z) + bk(y, z)$
- positive-definiteness, $k(x, x) > 0$ for x non-zero

This must hold for all elements $x, y, z \in V$ of the vector space and all scalars $a, b \in \mathbb{R}$. It is easy to see that the dot product from the previous section is in fact an inner product. For the modification (4.2), this requires a "change of perspective": As a function of the measured intensities, it is not an inner product as soon as φ is not linear; for example, doubling all intensities of one spectrum multiplies the score by $\sqrt{2}$ instead of 2 for $\varphi(x) = \sqrt{x}$. But if we pretend that we have transformed both spectra first, replacing every intensity x_m by $\sqrt{\psi(m)}\,\varphi(x_m)$, then it is simply the dot product of the transformed spectra (for $\psi(m) > 0$). Usually, it is not a smart idea to modify the experimental data; but here, we have to pretend that we did, if we want to inherit the mathematical elegance of inner products.

I do not want to go into the details of what a *vector space* is; but mass spectra turn out to be a good example of a vector space, because it is clear how to add two spectra, and how to multiply a spectrum by a scalar (real number, multiply intensities). Inner products are usually denoted $\langle x, y \rangle$ instead of $k(x, y)$ in math, but I will stick with the notation "$k(x, y)$" that many readers are probably more familiar with. Personally, I much prefer the term "scalar product" for an inner product, because this directly tells you that the result of a scalar product is a scalar, i.e., a real-valued number. You can also define scalar products over the complex numbers, but we do not need that here. Many people use the terms "inner product" and "dot product" interchangeably, even for inner products that cannot be written as a product $x^T y$ of two vectors x, y; with "scalar product", this confusion would not arise.

From the definitions of inner products — only those are needed and nothing more — we can infer a few more pleasant properties:

- $k(x, \mathbf{0}) = k(\mathbf{0}, y) = 0$ where $\mathbf{0}$ denotes the zero vector (in our case, the empty spectrum)
- $k(x, x) = 0$ if and only if $x = \mathbf{0}$
- $k(x, ay + bz) = ak(x, y) + bk(x, z)$

Again, this holds for all elements $x, y, z \in V$ of the vector space, and all scalars $a, b \in \mathbb{R}$. Potentially the best thing for us is that the inner product allows us to define a *norm* $\|x\| := \sqrt{k(x, x)}$. We can do so for any inner product k. And, this norm automatically fulfills beautiful properties such as $\|ax\| = |a| \cdot \|x\|$, the *triangle inequality* $\|x + y\| \leq \|x\| + \|y\|$, and the *Cauchy-Schwarz inequality*

$$|k(x, y)| \leq \|x\| \cdot \|y\| \,. \tag{4.3}$$

This holds for all $x, y \in V$ and all $a \in \mathbb{R}$. All we need to show those properties, is the definition of the inner product; no strings attached. In particular, all of the above holds for the dot product $x^T y$ from the previous section.

Inequality (4.3) is of particular interest for us, since it allows us to define a sensible measure of similarity between two spectra: We define the *normalized similarity* of $x, y \neq \mathbf{0}$ as $k(x,y)/(\|x\| \cdot \|y\|)$; by (4.3), its absolute value is at most one. For the dot product, this is exactly the cosine score (4.1).

Beyond the many mathematical amenities of using inner products and norms, using an inner product also has the advantage that it makes our similarity calculations intuitive: If we double peak intensities, we expect the similarity value to double; otherwise, the idea of spectrum normalization is bewildering. If we add two spectra, then the two similarities should add up, too. Our normalized similarity is always between 0 and 1. A spectrum x has normalized similarity 1 to itself; furthermore, no other spectrum y can be more similar to x than the spectrum x itself.

4.3 Mass thresholding: Not an inner product

Let us assume that both our query and reference are high mass accuracy data; how can we generalize the ideas from the previous section? Simply forgetting about the fact that our data is high mass accuracy, is the easiest (and dumbest) thing we can do; can we do better?

We are given a mass spectrum with peak masses $m_1 < \cdots < m_n$ and corresponding intensities $h_1, \ldots, h_n$. Recall that it is usually not a good idea to interact with the "raw data" that comes out of the machine; instead, we assume that peak picking (centroiding) has been performed on the spectrum. Note that a peak list is only part of what was detected in the spectrum: The instrument software and the peak picking algorithm decide which peaks are reported, and how many. Just because we were not involved in this decision, does not mean that it is God-given; see also the last paragraph of Sec. 13.1. Let $m'_1, \ldots, m'_{n'}$ and $h'_1, \ldots, h'_{n'}$ be the peak masses and intensities, respectively, of a second spectrum. Furthermore, let us assume that we are given a mass accuracy $\delta > 0$. Keep in mind that we are using absolute mass accuracies solely for the sake of presentation; it is straightforward how to take into account relative mass accuracy, or (best) a combination of both.

Now, a straightforward way to proceed is as follows: When calculating the similarity of the two spectra, we sum up only those $h_i h'_j$ that satisfy $|m_i - m'_j| \leq \delta$. This appears to be the exact analogue of what we have been doing for unit mass accuracy. Unfortunately, it is not: There might be multiple peaks in one spectrum that fit the other spectrum; worse, there may be peak masses m_i, m_{i+1} in one of the spectra such that $m_{i+1} - m_i \leq \delta$. We will now discuss numerous ways of dealing with this situation; unfortunately, all of them result in counterintuitive behavior, and none results in an inner product.

For our examples, consider a first spectrum with two peaks $(m_1, h_1) = (100, 1)$ and $(m_2, h_2) = (102, 1)$ and a second spectrum with one peak $(m'_1, h'_1) = (101, 1)$. Furthermore, assume $\delta = 1$; recall that we are using integer masses solely for the sake of easier presentation. From what we have said above, it is clear that peaks at masses 100 and 102 cannot match. Consequently, the similarity of the first spectrum to itself must be 2, and its "norm" is $\sqrt{2}$. Clearly, the similarity of the second spectrum to itself is 1, and so is its "norm". If we assume an all-to-all matching between peaks, then the similarity between the two spectra is 2, which is larger than the product of the two "norms". After "normalization", the second spectrum is more similar to the first than to itself. We could demand that only a single peak from the first spectrum may be matched to any peak of the second spectrum; unfortunately, that makes our similarity measure asymmetric.

Next, we could demand that at most one peak of the first spectrum is matched to at most one peak in the second spectrum; we again add up products of peak intensities, and find the assignment (matching) of peaks that maximizes the similarity. This would at least guarantee again that the similarity of two "normalized" spectra is at most 1. Luckily, computing optimal *matchings* is a well-researched problem in theoretical computer science. Nevertheless, computing a matching is substantially more demanding than all other algorithms in this chapter. In addition, this may result in counterintuitive peak matchings where

the order of peaks in the two spectra is not respected: Peak m_i may be matched to m'_{j+1} and simultaneously, m_{i+1} might be matched to m'_j. This clearly violates our chemical and physical intuition. A better way to find a matching between peak pairs, is to *align* the two spectra as sequences, see Sec. 4.5. This gets rid of the "peak order flipping" issue mentioned above and, at the same time, substantially reduces computation time. Yet, even this approach does not result in an inner product: Stick ten peaks of intensity one into a mass interval of width δ, then increase the first peak in one spectrum and the last peak in the other spectrum to intensity 100. The best matching that respects peak mass monotonicity, will now match the two peaks with intensity 100 and ignore all others: Bilinearity would result in score 10009, but the alignment score is 10000. Hence, the similarity measure is not linear.

You may think about other similarity measures, but to the best of my knowledge, nobody was ever able to develop a measure that is an inner product. I conjecture that under mild assumptions, one can even prove that no non-trivial inner product exists. So, what can we do to nevertheless compute a similarity measure that behaves reasonably, that is, like an inner product?

Well, ignorance is bliss. In practice, most methods simply ignore the problem and use one of the above similarity measures anyway — usually with reasonable results. But for high mass accuracy data, we can do better, see the next section.

4.4 The probability product kernel: An inner product at last

The following is for high mass accuracy data. It is most useful when the reference spectrum has peak intensities, but can also be applied for barcode spectra. It is a many-to-many scoring, where every peak in one spectrum is scored against every peak in the other.

We now generalize the ideas of Sec. 4.1 for spectra with high mass accuracy. Given our failures in the previous section, it is surprising how simply one can come up with an inner product which is mathematically elegant, but also very fast to compute in practice. On the downside, we need more math than usual in this textbook. A straightforward generalization of the dot product of two vectors of finite length is the *dot product of two functions* $g, g' : \mathbb{R} \to \mathbb{R}$,

$$k(g, g') = \int_a^b g(x) \cdot g'(x)\, dx \tag{4.4}$$

where $a < b$. The term "dot product" is in order as we are multiplying the two functions. This is known to be an inner product as long as all integrals (that is, their values) are finite. Conceptually, it is a direct generalization of the dot product for vectors, replacing the sum by an integral. Note that g' denotes a second function, *not* the derivative of g. Also note that $a = -\infty$ and/or $b = +\infty$ may hold as long as, again, all integrals are finite.

To ensure values between 0 and 1, we can again normalize using the norms of the two functions,

$$\frac{k(g, g')}{\|g\|\,\|g'\|} \quad \text{where} \quad \|g\| = \sqrt{k(g, g)}.$$

We may apply this to the raw (not peak-picked) spectra that are measured by the instrument, but that is not a good idea for many reasons: To name one, this will put a lot of weight on the baseline of the mass spectrum, something we definitely do not want. (In reality, raw spectra are discrete signals, but it is straightforward how to treat these as continuous functions.)

A better idea is to use peak lists as in the rest of this book, and to generate an idealized spectrum from the peak list, being the sum of Gaussians at the appropriate positions and with the appropriate intensities. We use some standard deviation σ roughly corresponding to the mass accuracy of the measurement, and define

$$f(x) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right)$$

to be the Gaussian curve. This happens to be the probability density of the normal distribution with mean zero and variance σ^2; but we will *not* use this interpretation here. Note that $\int_{\mathbb{R}} \exp(-x^2)\ dx = \sqrt{\pi}$ and, consequently, $\int_{\mathbb{R}} f(x)\ dx = 1$; this is a famous equation, and basically every math student has seen a proof during his or her studies. Hence, we may set $a = -\infty$ and $b = +\infty$ in (4.4). We may use other "peak shapes" but the Gaussian appears to be a somewhat natural choice: Even though calculations for mass spectrometry physics tell us that measured peaks should have shapes slightly different from a Gaussian function, it is still common to model them as Gaussians. Even more important is the observation that mass deviations follow roughly a normal distribution, see Sec. 4.7. Now, the complete *ideal spectrum* is

$$g(x) := \sum_i h_i f(x - m_i)$$

where $m_1, \ldots, m_n$ are the peak masses and $h_1, \ldots, h_n$ the corresponding intensities. Given a second ideal spectrum $g'(x) := \sum_j h'_j f(x - m'_j)$ for peak masses $m'_1, \ldots, m'_{n'}$ and intensities $h'_1, \ldots, h'_{n'}$, we have

$$k(g, g') = k\left(\sum_i h_i f_{m_i}, \sum_j h'_j f_{m'_j}\right) = \sum_{i,j} h_i h'_j \cdot k(f_{m_i}, f_{m'_j}) \tag{4.5}$$

where $f_m(x) := f(x - m)$, because k is an inner product.

It remains to compute $k(f_m, f_{m'})$. But instead, we first compute $k(f, f_\delta)$ for $\delta = m' - m$. Now,

$$\begin{aligned}
\int f(x) \cdot f_\delta(x)\ dx &= \int f(x) \cdot f(x - \delta)\ dx \\
&= \int \frac{1}{2\pi\sigma^2} \exp\left(\frac{-x^2 - (x-\delta)^2}{2\sigma^2}\right) dx \\
&= \frac{1}{2\pi\sigma^2} \int \exp\left(\frac{-2x^2 + 2\delta x - \delta^2}{2\sigma^2}\right) dx \\
&= \frac{1}{2\pi\sigma^2} \int \exp\left(\frac{-x^2 + \delta x - \frac{1}{2}\delta^2}{\sigma^2}\right) dx \\
&= \frac{1}{2\pi\sigma^2} \int \exp\left(\frac{-(x - \frac{1}{2}\delta)^2 + \frac{1}{4}\delta^2 - \frac{1}{2}\delta^2}{\sigma^2}\right) dx \\
&= \frac{1}{2\pi\sigma^2} \cdot \exp\left(\frac{-\frac{1}{4}\delta^2}{\sigma^2}\right) \cdot \int \exp\left(\frac{-(x - \frac{1}{2}\delta)^2}{\sigma^2}\right) dx \\
&= \frac{\sqrt{\pi\sigma^2}}{2\pi\sigma^2} \cdot \exp\left(\frac{-\delta^2}{4\sigma^2}\right) \cdot \frac{1}{\sqrt{2\pi(\sigma/\sqrt{2})^2}} \int \exp\left(\frac{-(x - \frac{1}{2}\delta)^2}{2(\sigma/\sqrt{2})^2}\right) dx
\end{aligned} \tag{4.6}$$

The last integral is one, as we integrate the density of a normal distribution over $\mathbb{R}$. Hence,

$$k(f, f_\delta) = \int_{-\infty}^{+\infty} f(x) f_\delta(x)\ dx = \frac{1}{\sqrt{2\pi \cdot 2\sigma^2}} \cdot \exp\left(-\frac{\delta^2}{2 \cdot 2\sigma^2}\right). \tag{4.7}$$

Now,

$$k(f_m, f_{m'}) = k(f, f_\delta)$$

with $\delta = m' - m$, which is what we wanted to compute.

Eq. (4.7) is just the probability density function of the normal distribution (Gaussian function) with mean 0 and variance $2\sigma^2$. Nevertheless, this is *not* the probability that mass difference δ is observed: The probability for observing mass deviation exactly δ is zero, for any δ. This simply follows from the fact that the normal distribution is a continuous distribution. In this sense, the dot product does not have a "clean" stochastic interpretation, see Sec. 4.7 below where we will use the cumulative distribution function of the

normal distribution. On the other hand, we can interpret this value as being proportional to the probability that the mass deviation is "δ or sufficiently close". This gives reasonable values as the normal distribution is "well-behaved" (unimodal, symmetric, differentiable, not heavy-tailed etc.).

Combining (4.5) and (4.7) we reach

$$k(g,g') = \int_{-\infty}^{\infty} g(x) \cdot g'(x)\, dx = \frac{1}{\sqrt{2\pi \cdot 2\sigma^2}} \cdot \sum_{i,j} h_i h'_j \cdot \exp\left(-\frac{(m_i - m'_j)^2}{2 \cdot 2\sigma^2}\right) \tag{4.8}$$

which allows us to compute the inner product in time $O(n \cdot n')$. There is no need to do any numerical integration of the two functions; all we need is to invoke the exponential function $n \cdot n'$ times.

But in practice, we can bring down the running time to linear if σ is chosen reasonably small: Assume $|m - m'| \geq 10\sigma$ holds.[1] Then,

$$\exp\left(-\frac{(m-m')^2}{2 \cdot 2\sigma^2}\right) \leq \exp\left(-\frac{(10\sigma)^2}{2 \cdot 2\sigma^2}\right) = \exp(-25) = 1.39 \cdot 10^{-11}$$

and we can safely ignore the corresponding summand even if peak intensities are high, as it will not contribute substantially to the dot product. For comparison, $|m - m'| \geq 8\sigma$ only guarantees values below $1.13 \cdot 10^{-7}$, and $|m - m'| \geq 6\sigma$ values below $1.23 \cdot 10^{-4}$. (In case all summands in (4.8) are small and this summand has a noteworthy effect on the sum, then the two spectra are so dissimilar that, again, it does not make a difference if we consider the summand or not.) Using a sweep over both sorted peak lists, we can compute the sum in time $O(n + n' + P)$, where P is the number of peak pairs (i, j) with $\left|m_i - m'_j\right| < 10\sigma$; this is linear if every peak has only a constant number of partners within 10σ. In particular, these considerations allow us to compute the norm of g in linear time if the mass difference between consecutive peaks is at least 10σ: Then,

$$\|g\| = \sqrt{k(g,g)} \approx (4\pi\sigma^2)^{-1/4} \sqrt{\sum h_i^2}$$

which, up to a constant factor that cancels in the normalized score, is the same as for the unit mass accuracy case.

Eq. (4.8), being a dot product, is a kernel function. We do not want to go into the mathematical details here, but such kernels are of importance in machine learning, namely for kernel methods such as the well-known support vector machines. Notably, a different kernel, where intensities are not weights but a second coordinate of each peak, is frequently used for that purpose [149]: The *probability product kernel* is defined as

$$k(g,g') = \frac{1}{nn'} \frac{1}{4\pi \cdot \sigma_{\text{mass}} \cdot \sigma_{\text{int}}} \cdot \sum_{i,j} \exp\left(-\frac{(m_i - m'_j)^2}{2 \cdot 2\sigma_{\text{mass}}^2} - \frac{(h_i - h'_j)^2}{2 \cdot 2\sigma_{\text{int}}^2}\right) \tag{4.9}$$

where $\sigma_{\text{mass}}, \sigma_{\text{int}}$ are standard deviations of mass and intensity, respectively. In machine learning, these (hyper)parameters are usually determined using cross-validation. Eq. (4.9) assumes zero covariance between masses and intensities — or, more precisely, between mass deviations and intensity deviations.

Despite its mathematical elegance, I have to say that neither (4.5) nor (4.9) is frequently used for database searching: Users apparently dislike the fact that in these dot products, both peak intensities *and* mass deviations are scored simultaneously. As a result, even a seemingly perfect fit between two spectra may get a dot product of only 0.7, because the mass deviations between peaks are a little larger than expected. From the user perspective, the scores are not a generalization of the dot product for unit mass accuracy, but rather different.

[1] The following considerations are *not connected* to the 5 sigma rule from physics or the 6 sigma rule from quality management: Those rules consider the cumulative distribution of the normal distribution, whereas we are using its probability density function.

4.5 Matching spectra and the peak counting score for high mass accuracy

In this section, we assume that $\mathcal{M}$ is the *reference* we want to compare against; and that $\mathcal{M}'$ is the *query* of our sample using an MS instrument. For the ease of presentation, we assume that both $\mathcal{M}$ and $\mathcal{M}'$ are sets of masses. In fact, we can easily add more "peak attributes" to this framework without having to change the formal presentation: We can think of these attributes as maps from the set of masses, to some set representing the possible attribute states. One such attribute that we will make use of repeatedly, is peak intensities in the measured spectrum. For the reference spectrum, a possible peak attribute is the ion series the peak stems from.

The following is for high mass accuracy data. It assumes that both spectra are barcode spectra, for example after intensity thresholding of the query. It is a one-to-one scoring, where each peak in one spectrum is scored against at most one peak in the other.

Given a protein string, it is quite easy to simulate, say, tryptic digestion *in silico*, see Exercise 1.1. But it is similarly easy to simulate the tandem mass spectrum of a peptide — at least, if we assume some simple model of peptide fragmentation, see Chapter 2, in particular Sec. 2.6. In fact, we have implicitly "simulated" such spectra there. We leave the details to the reader, see Exercises 4.3 and 4.10.

In Chapter 2, we have implicitly introduced a simple approach to compare two mass spectra: We did so by counting the peaks that occur both in the measured spectrum $\mathcal{M}'$ and in the reference spectrum $\mathcal{M}$. This number will be called *peak counting score* in the following, but goes under many different names in the literature. The idea behind this, is that the measured spectrum is fixed, whereas we are searching for a best match in the database. As mentioned repeatedly, we have to allow for some mass deviation $\varepsilon > 0$ between the masses of measured and reference peak. In the following, we will also look at other ways to compute a *score* for the reference spectrum $\mathcal{M}$ by comparing it to the fixed measured spectrum $\mathcal{M}'$.

What exactly do we mean with "counting common peaks"? In fact, there are at least four different interpretations:

1. We want to match pairs of peaks: That is, every peak in the reference spectrum $\mathcal{M}$ can be matched with at most one peak in the measured spectrum $\mathcal{M}'$, and vice versa, to contribute toward the score.

2. Each peak in the reference spectrum $\mathcal{M}$ can be matched with at most one peak in the measured spectrum $\mathcal{M}'$; but a peak in the measured spectrum $\mathcal{M}'$ may be matched to many peaks in the reference spectrum $\mathcal{M}$.

3. Each peak in the measured spectrum $\mathcal{M}'$ can be matched with at most one peak in the reference spectrum $\mathcal{M}$; but a peak in the reference spectrum $\mathcal{M}$ may be matched to many peaks in the measured spectrum $\mathcal{M}'$.

4. Each peak in the reference spectrum $\mathcal{M}$ may be matched with every peak in the measured spectrum $\mathcal{M}'$, and every such pair contributes toward the score.

Intuitively, the first interpretation appears to be the most "natural"; but it turns out that the second interpretation is also quite reasonable in many applications. We will call the first a *one-to-one matching*, and the second a *one-to-many* matching. In contrast, the third interpretation should hardly ever be relevant in applications. The fourth interpretation is a *many-to-many* matching; this is exactly what the inner products in Sec. 4.4 do, where the contribution of a pair decreases with its mass difference. Here, we will not cover many-to-many matchings again but just mention in passing that all algorithms from the previous section can also be applied with a different mass scoring function, see below. For the moment, we concentrate on the one-to-one matching case.

Example 4.1. We now give an example meant to demonstrate various problems of the peak counting score. Assume that we have measured the spectrum

$$\mathcal{M}' = \{200, 300, 500, 515, 700\}$$

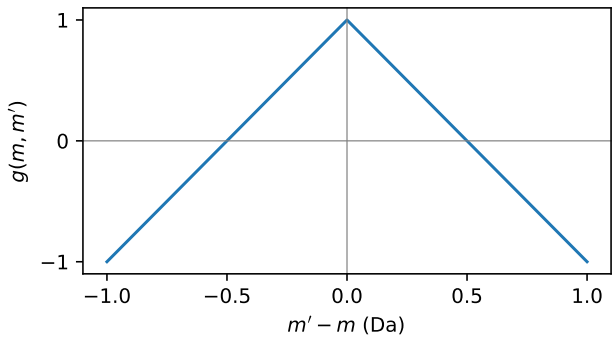


Figure 4.1: The mass scoring function $g(m, m') = 1 - 2\,|m - m'|$ from Example 4.2, as a function of the mass difference $m' - m$.

and we want to compare it against a set of reference mass spectra in our database, namely:

$$\begin{aligned}
\mathcal{M}_1 &= \{100, 175, 350, 480, 490, 550\} \\
\mathcal{M}_2 &= \{200, 270, 300, 500\} \\
\mathcal{M}_3 &= \{205, 505, 705, 850\} \\
\mathcal{M}_4 &= \{190, 310, 490, 710\} \\
\mathcal{M}_5 &= \{100, 150, 200, 250, \ldots, 600, 650, 700\}
\end{aligned}$$

Assume that $\varepsilon = 10$ is the mass deviation that we believe to be reasonable. Now, we find that $\mathcal{M}_1$ has one peak in common with $\mathcal{M}'$; both $\mathcal{M}_2$ and $\mathcal{M}_3$ have three; and both $\mathcal{M}_4$ and $\mathcal{M}_5$ have four peaks in common.

What are the problems with the peak counting score in Example 4.1? First, changing parameter ε slightly can dramatically change the score. For example, spectrum $\mathcal{M}_4$ has score four for $\varepsilon = 10$, but if we instead choose $\varepsilon = 9.9$ then the peak counting score decreases to zero. But also for spectra that are not in this "critical zone", it is understood that $\mathcal{M}_2$ fits the observed data better than $\mathcal{M}_3$, but this is not reflected in the score. Finally, reference spectra with many peaks such as $\mathcal{M}_5$ get a better score, because they are more likely to hit a peak mass in $\mathcal{M}'$ just by chance. We have observed this problem at the end of Sec. 2.5.3.

In view of this, it seems reasonable to score mass deviations a little more carefully. To this end, we assume that we are given some *mass scoring function* $f : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ that, for a pair of peaks at masses m (for the reference peak) and m' (for the measured peak), judges the similarity of these peaks based on their masses. For a mass scoring function to be true to the application, we may demand an additional property: If $M' < m' < m$ or $m < m' < M'$ holds, then $f(m, M') < f(m, m')$. Similarly, if $M < m < m'$ or $m' < m < M$ then $f(M, m') < f(m, m')$. A mass scoring function is called *strictly monotone* if it satisfies these two conditions. A weaker condition is that $M' < m' < m$ or $m < m' < M'$ implies $f(m, M') \leq f(m, m')$, and that $M < m < m'$ or $m' < m < M$ implies $f(M, m') \leq f(m, m')$. In this case, the mass scoring function is called *monotone*. For example, the peak counting score for any $\varepsilon > 0$ is monotone but not strictly monotone. The above are quite reasonable conditions: For example, $f(M, m') > f(m, m')$ for $M < m < m'$ or $m' < m < M$ would imply that matching the measured peak at mass m' with the more distant reference peak M, is more sensible than matching it with the closer reference peak m.

Example 4.2. Let $g(m, m') := 1 - 2\,|m - m'|$ for $m, m' \in \mathbb{R}_{\geq 0}$. Then, g is a mass scoring function that is strictly monotone. In particular, we have $g(m, m') \leq 1$ for all m, m'; $g(m, m') = 1$ if and only if $m = m'$; and $g(m, m') = 0$ for $|m - m'| = \frac{1}{2}$. See Fig. 4.1.

We now assume that peak pairs are scored by some score function score : $\mathcal{M} \times \mathcal{M}' \to \mathbb{R}$. Such a score function is usually derived from a mass scoring function, but can take into account other attributes such as

intensities. An alignment of the mass spectra $\mathcal{M}$ and $\mathcal{M}'$ is a *matching* of the two sets, where a subset of $\mathcal{M}$ is bijectively mapped onto a subset of $\mathcal{M}'$. To penalize peaks that are not matched with any counterpart, we introduce a gap character -, and we extend the domain of score to also cover the gap character. Here, $\text{score}(\text{-}, m') \leq 0$ penalizes an additional peak $m' \in \mathcal{M}'$, whereas $\text{score}(m, \text{-}) \leq 0$ penalizes a missing peak $m \in \mathcal{M}$. We define the score of a matching as:

$$\sum_{m \text{ matches } m'} \text{score}(m, m') + \sum_{\text{additional peaks } m' \in \mathcal{M}'} \text{score}(\text{-}, m') + \sum_{\text{missing peaks } m \in \mathcal{M}} \text{score}(m, \text{-}) \tag{4.10}$$

Let $\mathcal{A} \subseteq \mathcal{M} \times \mathcal{M}'$ be the matching of an alignment of $\mathcal{M}, \mathcal{M}'$; we write $\mathcal{A}$ as in "alignment". We say that the alignment is *crossing* if there exist matched mass pairs (m_1, m_1') and (m_2, m_2') in $\mathcal{A}$ such that $m_1 < m_2$ but $m_1' > m_2'$ holds, or $m_1 > m_2$ but $m_1' < m_2'$ holds. Otherwise, the alignment is *crossing-free*.

Example 4.3. Let $\mathcal{M} = \{500.3, 500.5\}$ and $\mathcal{M}' = \{500.0, 500.4\}$, and let us use the mass scoring function g from Example 4.2. The matching $\{(500.3, 500.0), (500.5, 500.4)\}$ is crossing-free and has score $0.4 + 0.8 = 1.2$. In contrast, the matching $\{(500.3, 500.4), (500.5, 500.0)\}$ is crossing, and has score $0.8 + 0 = 0.8$.

Crossing matchings such as the second one in Example 4.3 are physical nonsense: There is no physical process under which two peaks in a mass spectrum would switch order. Mass errors are small and, locally, the measured mass is an increasing function of the true mass. So, we only consider crossing-free matchings. The optimal crossing-free matching can be found by aligning the spectra, so we use dynamic programming for the table $D[0 \ldots n, 0 \ldots n']$ with $n := |\mathcal{M}|$ and $n' := |\mathcal{M}'|$. We initialize $D[0,0] = 0$, $D[i,0] = D[i-1,0] + \text{score}(m_i, \text{-})$ for $i = 1, \ldots, n$, and $D[0,j] = D[0,j-1] + \text{score}(\text{-}, m_j')$ for $j = 1, \ldots, n'$. We use the following recurrence to fill the table:

$$D[i,j] = \max \begin{cases} D[i-1,j] + \text{score}(m_i, \text{-}) \\ D[i-1,j-1] + \text{score}(m_i, m_j') \\ D[i,j-1] + \text{score}(\text{-}, m_j') \end{cases} \tag{4.11}$$

After filling the matrix, the optimal score can be found in entry $D[n,n']$. To find the optimal alignment we use backtracking through D. Obviously, the method requires $O(n \cdot n')$ time and memory. Consider the measured spectrum $\mathcal{M}'$ and reference spectrum $\mathcal{M}_2$ from Example 4.1: Using the mass scoring function from Example 4.2 with gap penalty -1, the best alignment has score 0. In application, the optimal alignment can usually be found much faster than the worst-case running time suggests: If $\text{score}(m, m') < \text{score}(m, \text{-}) + \text{score}(\text{-}, m')$, then m and m' are never matched in an optimal alignment, as replacing this pair by two gaps increases the score. For a monotone mass scoring function and gap penalties that do not depend on the peak, the candidate partners of each peak m_i form a contiguous block in $\mathcal{M}'$, and these blocks move monotonically with i, see Exercise 4.8. We find all K candidate pairs in time $O(n+n'+K)$ by sweeping over both sorted lists. The score of any alignment equals the score of aligning all peaks with gaps, plus $w(i,j) := \text{score}(m_i, m_j') - \text{score}(m_i, \text{-}) - \text{score}(\text{-}, m_j')$ for every matched pair (i,j). Hence, we search for a crossing-free matching of candidate pairs with maximum total weight. We process the blocks in order of i; columns left of the current block are never touched again, so we keep their best value in a single variable. This requires $O(n + n' + K)$ time and memory. Without monotone blocks, one can still restrict the computation to the candidate pairs using sparse dynamic programming [105], at the price of a small extra factor. For high mass accuracy data, K is small: Mass deviations are measured in ppm, and a fragmentation spectrum with a hundred peaks between 0 and 1000 Da is mostly empty. Many peaks within a few ppm would mean that the compound has many fragments of almost identical mass, which is extremely unlikely.

Claude helped!

4.6 Additional and missing peaks: To score or not to score?

If our reference is a measured spectrum, we may indeed penalize both *additional* peaks, which are found in the query but cannot be matched to any peak of the reference, and *missing* peaks, which are found in

the reference but not in the query. But is this always the case? The following is a somewhat philosophical reflection on the cases and applications where this makes sense, and on those where it does not.

Consider additional peaks first, for a simulated reference spectrum: Our simulation may be incomplete, not covering all, say, fragmentation reactions. Maybe, these fragmentation reactions have never been described in the literature. Maybe, even all available data will not allow us to explain this peak. Also, there is the chance that "impurities" will screw up our ranking.

Things are even worse for missing peaks: How sure are we that the peak should indeed be observable in the query spectrum? In case our query was intensity-thresholded, we have to be very cautious — potentially, the query spectrum does contain the peak, but it is so small that it was missed. With intensities, a small peak is lost more easily than a high-intensity peak; dot products, by design, put more weight (attention) on high-intensity peaks. Hence, for simulated references we usually do not penalize additional peaks; for missing peaks, see Sec. 4.10.

4.7 Stochastic model for scoring mass deviations

Let $m_1, \ldots, m_n$ be the peak masses of the reference spectrum (our hypothetical "truth"), and $m'_1, \ldots, m'_{n'}$ of the query spectrum. We assume that the matching $\mathcal{A}$ between peaks from the two spectra is already established, and $(i, j) \in \mathcal{A}$ means that we match peak i from the reference with peak j from the query. In practice, this matching is computed by the alignment from Sec. 4.5, using the score derived in this section as the pair score score.

For our score, we will use log-odds scores, as defined in statistics: We want to differentiate between two statistical models, one for our hypothesis and one for the background. Here, we look at a pair of peaks, one from the measured spectrum and one from the reference spectrum, that have been matched by our spectrum alignment algorithms. Now, the two models are "the measured peak is a realization of the reference peak" vs. "the measured peak is simply noise, and has nothing to do with the reference peak."

Odds are used to differentiate between the two models, by computing the ratio

$$\textit{odd score} = \frac{\mathbb{P}(D|H_1)}{\mathbb{P}(D|H_0)}$$

where D is the observed data (the peak in the measured spectrum), H_1 is our hypothesis (the measured peak belongs to the reference peak), and H_0 is the null model (the measured peak is noise). For $\textit{odd score} > 1$ we would accept the model H_1, and for $\textit{odd score} < 1$ the null model H_0 is more likely.

Log-odds scores do pretty much the same as odds:

$$\textit{log odd score} = \log \frac{\mathbb{P}(D|H_1)}{\mathbb{P}(D|H_0)} \tag{4.12}$$

Here, the logarithm can be computed to an arbitrary (but fixed) base, such as the natural logarithm with base e. For $\log_2$ the resulting log-odds scores are called *bit scores*. For $\textit{log odd score} > 0$ we accept the model, for $\textit{log odd score} < 0$ we reject it. Log-odds scores have the advantage that we can sum them, instead of multiplying likelihoods, to obtain a statistically meaningful number: That is, the log-odds score that all the matched peaks of the measured spectrum belong to their reference counterparts, vs. all the measured peaks are noise.

Now, assume that the model is true, that is, the measured peak belongs to the matched reference peak. Then, it is usually hard to predict the intensity of a simulated reference peak. But we can use the difference between the measured mass and the reference mass to assess whether the model holds: We want to assess the probability that the mass differences between measured and reference peaks, corresponding to the *mass error* of the measurement, can get this large or larger by chance.

The probability to observe a certain mass error, clearly depends on the accuracy of the instrument: If the instrument has a bad mass accuracy (for example, ion trap MS) then we will observe large mass errors much more often than for an instrument with excellent mass accuracy, such as Orbitrap MS. In fact,

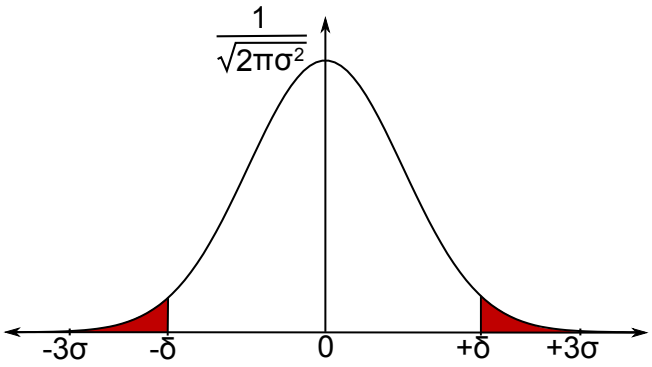


Figure 4.2: We model the distribution of mass deviation δ as a normal distribution $\mathcal{N}(0, \sigma^2)$ with mean zero. Then, 99.7 % of the measured mass deviations lie between -3σ and $+3\sigma$.

mass spectrometry literature reports *mass accuracies* of instruments and measurements: This is a unit-free number, usually given in *parts per million* (ppm), showing the relative mass accuracy of the measurement or instrument. For example, if we measure an ion with mass 1000 Da at mass 1000.03 Da, then the "mass accuracy" of the measurement is

$$\frac{|1000 - 1000.03|}{1000} = 3 \cdot 10^{-5} = 30\,\text{ppm}. \tag{4.13}$$

Unfortunately, the mass accuracy reported in the literature often refers to such a single mass difference: Zubarev and Mann [379] coined the term *anecdotal mass accuracy* for the "selective reporting of mass measurements, usually to demonstrate the capabilities of the author's instrument." Such anecdotal mass accuracy "should clearly be distinguished from routine instrument performance in day to day use." Zubarev and Mann also proposed to use the term "*mass deviation*" instead of "mass accuracy" for an individual mass error, such as the one in (4.13).

We need, in contrast, a statistical mass accuracy that assigns probabilities to different mass errors. It turns out that mass errors are roughly normally distributed with mean zero. By the central limit theorem, a measurement error that is the sum of many small, independent contributions is approximately normally distributed. But this fact has also been verified experimentally in at least two studies [162, 379].

But before we continue, a *warning* is in order: Observed mass errors are in fact the sum of a systematic mass error due to poor calibration, and the statistical mass error introduced above. The systematic mass error can be countered by calibration using (internal or external) standards, or by hypothesis-driven recalibration, see Sec. 4.12 below for more details. We first have to remove the systematic mass error from the measurement; only then is it reasonable to assume that mass errors are statistically distributed.

The probability $\mathbb{P}(\text{absolute mass deviation} \geq \delta)$ is the integral over both tails, labeled red in Figure 4.2. The expectation value is $\mu = 0$ for a well-calibrated instrument. The standard deviation is $\sigma = \frac{1}{3}\frac{\alpha}{10^6}m$, where α is the mass accuracy of the instrument in ppm. We assume that 99.7 % of the measurements lie within the stated mass accuracy; this is consistent with $\pm 3\sigma$. Note that mass accuracies are sometimes reported as RMS error, which corresponds to 1σ; check what the specification of your instrument means. Using the normal distribution, we cannot rule out that arbitrarily large mass deviations occur; we just assume that they are arbitrarily unlikely. For example, we implicitly assume that 99.9999998 % of all measurements are at least within twice the stated mass accuracy, and less than two in a billion measurements show a larger mass deviation.[2] We estimate the probability of observing a mass difference of $|m - m'|$ or larger as:

$$\mathbb{P}(D|H_1) = \mathbb{P}(\text{mass difference of } |m - m'| \text{ or more}) = \frac{2}{\sqrt{2\pi}} \int_z^\infty e^{-t^2/2} dt = \operatorname{erfc}\left(\frac{|m - m'|}{\sqrt{2}\,\sigma_{\text{mass}}}\right) \tag{4.14}$$

[2] We cannot rule out that a meteor hits and destroys the earth tomorrow; it is just very, very unlikely.

with $z := \frac{|m-m'|}{\sigma_{\text{mass}}}$, where m and m' are the masses of the reference and the measured peak, and σ_{mass} is the standard deviation of the Gaussian mass error distribution. The antiderivative of e^{-t^2} cannot be expressed by elementary functions. That is why practically all numerical libraries offer functions $\text{erf}(\cdot)$ and $\text{erfc}(\cdot)$ which allow us to compute approximations of excellent quality ("approximation" as in mathematical approximation theory, see Sec. 15.7). In particular, "erfc" denotes the *complementary error function* with $\text{erfc}(x) := \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\, dt$. Note that we use the probability of observing a mass difference this large or larger as a surrogate for $\mathbb{P}(D|H_1)$; strictly speaking, the likelihood of a continuous observation is a density, see Sec. 4.4. We will come back to the null model H_0 in Sec. 4.9.

The precursor ion can be scored in the same way: In a sense, it is one more peak, and its mass is usually measured in the MS1 spectrum, often with higher accuracy than fragment masses. For a candidate with theoretical precursor mass M and measured precursor mass M', we add $\log \text{erfc}\left(|M - M'| / (\sqrt{2}\, \sigma_{\text{prec}})\right)$ to the score. In practice, candidates are first filtered by precursor mass, see Sec. 12.10; the precursor term then ranks those candidates that pass the filter. Note that the precursor mass error is not independent of the fragment mass errors, as both are affected by the same calibration error, see Sec. 4.12.

Claude helped!

4.8 Stochastic model for scoring intensity deviations

Let $h_1, \ldots, h_n$ be the peak intensities of the reference spectrum, and $h'_1, \ldots, h'_{n'}$ of the query spectrum. Again, the matching $\mathcal{A}$ between peaks from the two spectra is already established, and $(i, j) \in \mathcal{A}$ means that we match peak i from the reference with peak j from the query.

First, the simplest way to compare intensities — which has been suggested repeatedly in the literature — is to use the absolute value of the difference, $|h - h'|$. Ignoring peak masses for a moment, this results in the sum of absolute differences,

$$\lambda \cdot \sum_{(i,j)\in\mathcal{A}} \left|h_i - h'_j\right|,$$

where the factor $\lambda > 0$ is required if you want to combine this score with other scores. Using the sum of absolute differences is more robust than the sum of squared differences, meaning that outliers have less impact. One may think that this approach, unlike others discussed here, does not have an underlying statistical model; but this is not the case. Let us assume that the absolute intensity deviations are exponentially distributed with parameter $\lambda > 0$; then, the probability to observe a deviation of $|h - h'|$ is $1-(1-\exp(-\lambda\,|h - h'|)) = \exp(-\lambda\,|h - h'|)$. (We use the cumulative distribution function to estimate the likelihood of a deviation of *at least* $|h - h'|$; but here, we could as well use the probability density function and end up with practically the same result.) Assuming independence, the likelihood of the data is the product of these likelihoods,

$$\mathbb{P}(h'_1, \ldots, h'_{n'} \mid h_1, \ldots, h_n, \lambda) = \prod_{(i,j)\in\mathcal{A}} \exp\left(-\lambda \cdot \left|h_i - h'_j\right|\right),$$

and instead of maximizing this likelihood we can as well maximize the log likelihood

$$\log \mathbb{P}(h'_1, \ldots, h'_{n'} \mid h_1, \ldots, h_n, \lambda) = \sum_{(i,j)\in\mathcal{A}} -\lambda \cdot \left|h_i - h'_j\right| = -\lambda \sum_{(i,j)\in\mathcal{A}} \left|h_i - h'_j\right|$$

which is in turn equivalent to minimizing $\sum_{(i,j)\in\mathcal{A}} \left|h_i - h'_j\right|$. Here, "log" denotes the natural logarithm. Hence, if we use the sum of absolute differences to decide on the best match, this is equivalent to a maximum likelihood approach *implicitly assuming that absolute deviations are exponentially distributed.* To combine this with other likelihood estimates, we still have to choose an appropriate $\lambda > 0$ as a multiplicative factor.

Second, the probability product kernel (4.9) explicitly models intensity deviations $h - h'$ using a normal distribution, but uses the probability density function instead of the cumulative distribution (which we prefer, see Sec. 4.7). Again, doing so has been suggested repeatedly in the literature.

Third, we can explicitly model the relative error h'/h of intensities. A somewhat natural choice is to model h'/h as a normal distribution with mean one, but this model has non-zero probability for negative values of h'/h, see below. Another possibility is to model h'/h by a log-normal distribution or, equivalently, $\log h'/h$ by a normal distribution with mean zero.

All of these approaches are not completely satisfactory in practice: Undisputedly, the intensity deviation has an absolute *and* a relative component. Can we model both error types simultaneously?

It turns out that this is indeed possible. We use a maximum likelihood estimator with two parameters, namely, absolute error $\sigma_{\text{abs}} > 0$ and relative error $\sigma_{\text{rel}} > 0$ of peak intensities. We statistically model the intensity error as

$$Y = x + D + E$$

where x is the expected (theoretical) intensity, Y is the random variable modeling the observed intensity, and D, E are random variables for relative and absolute noise, respectively. We assume that both relative noise $D \sim x \cdot \mathcal{N}(0, \sigma_{\text{rel}}^2) = \mathcal{N}(0, x^2\sigma_{\text{rel}}^2)$ and absolute noise $E \sim \mathcal{N}(0, \sigma_{\text{abs}}^2)$ are normally distributed: In detail, we assume that D, E have densities

$$f_D(\delta) = \frac{1}{\sqrt{2\pi x^2 \sigma_{\text{rel}}^2}} \exp\left(-\frac{\delta^2}{2x^2\sigma_{\text{rel}}^2}\right) \quad \text{and} \quad f_E(\epsilon) = \frac{1}{\sqrt{2\pi\sigma_{\text{abs}}^2}} \exp\left(-\frac{\epsilon^2}{2\sigma_{\text{abs}}^2}\right)$$

for $\delta, \epsilon \in \mathbb{R}$. (Warning: This ϵ is neither the empty string nor the mass accuracy, see below; but the Greek alphabet has only so many letters.) As in Sec. 4.4, we use the probability density function to estimate probabilities, see there for an explanation/excuse. The model can result in negative observed peak intensities, but this limitation is not relevant in practice, where relatively weak noise is observed.

We further assume that relative and absolute noise are independent. Given an observed intensity $y := h'$ and an expected intensity $x := h$, the likelihood of a decomposition $\delta + \epsilon = y - x$ of the observed deviation into relative and absolute noise is

$$\begin{aligned}\mathcal{L}_{y|x}(\delta, \epsilon) &= \frac{1}{\sqrt{2\pi}x\sigma_{\text{rel}}} \exp\left(\frac{-\delta^2}{2x^2\sigma_{\text{rel}}^2}\right) \cdot \frac{1}{\sqrt{2\pi}\sigma_{\text{abs}}} \exp\left(\frac{-\epsilon^2}{2\sigma_{\text{abs}}^2}\right) \\ &= \frac{1}{2\pi x\sigma_{\text{rel}}\sigma_{\text{abs}}} \exp\left(-\frac{\delta^2}{2x^2\sigma_{\text{rel}}^2} - \frac{\epsilon^2}{2\sigma_{\text{abs}}^2}\right).\end{aligned} \tag{4.15}$$

Minimizing $\frac{\delta^2}{x^2\sigma_{\text{rel}}^2} + \frac{\epsilon^2}{\sigma_{\text{abs}}^2}$ under the constraint $\delta + \epsilon = y - x$ gives $\delta_0 = (y-x)\frac{x^2\sigma_{\text{rel}}^2}{\sigma_{\text{abs}}^2 + x^2\sigma_{\text{rel}}^2}$ and $\epsilon_0 = y - x - \delta_0$; inserting, we find that the maximum likelihood is

$$\mathcal{L}_{y|x}(\delta_0, \epsilon_0) = \frac{1}{2\pi x\sigma_{\text{rel}}\sigma_{\text{abs}}} \exp\left(-\frac{(y-x)^2}{2(\sigma_{\text{abs}}^2 + x^2\sigma_{\text{rel}}^2)}\right). \tag{4.16}$$

We can see that δ_0, ϵ_0, corresponding to the maximum likelihood model, are not to be found on the right side of the equation. Hence, for the computation of the maximum likelihood, we actually do not have to compute δ_0 or ϵ_0 at all! Instead, we have the highly convenient situation that we can directly insert theoretical intensity $x = h$ and observed intensity $y = h'$ into equation (4.16) to estimate the maximum likelihood of the data. Note that (4.16) is very similar to the probability density function of the random variable $D + E \sim \mathcal{N}(0, \sigma_{\text{abs}}^2 + x^2\sigma_{\text{rel}}^2)$.

The above model is mathematically sound, but has a conceptual disadvantage when scoring several theoretical candidate spectra against one measured spectrum: The relative noise depends on the peak intensity in the candidate spectrum and, hence, each candidate spectrum is scored differently. Hence, we exchange the roles of x and y (expected/theoretical intensity vs. observed intensity) in our scoring. Likelihoods computed in this way are usually very large, positive values; we can instead use log odds, dividing values by the likelihood for some fixed δ_0, ϵ_0 such as $\delta_0 = 2y\sigma_{\text{rel}}$ and $\epsilon_0 = 2\sigma_{\text{abs}}$. Note that

this makes the scores of different candidates for the *same* measured spectrum comparable, but not the scores for *different* measured spectra: After exchanging roles, both the normalization and the relative error depend on the measured intensities. Hence, never use this score to compare candidates of different measured spectra, for example, to rank all hits of an LC-MS/MS run; see Chapter 5 for how to make scores comparable across spectra.

4.9 Stochastic model for scoring intensities against barcode peaks

We now complete the log-odds score from Sec. 4.7 for the case that the reference is a barcode spectrum, as is usually the case for simulated spectra. For a matched pair of peaks, the measured peak m' with intensity y is either signal from the reference peak m (hypothesis H_1), or it is noise (null model H_0). If m' is signal, then we score its mass deviation as in Sec. 4.7; its intensity tells us nothing, as the reference peak has no intensity to compare against.

For the null model, we cannot use the mass of the peak since, in general, noise peaks may appear at any mass. But we can use the peak intensity for this purpose: Evaluations have shown that noise peak intensities are roughly exponentially distributed; see for instance Fig. 4 in Goldberg *et al.* [127]. Let $\lambda e^{-\lambda x}$ be the density of the exponential distribution with parameter λ, where x is the peak intensity. The probability of observing a noise peak with intensity y or higher is

$$\mathbb{P}(\text{intensity noise} \geq y) = \int_y^\infty \lambda e^{-\lambda x} dx = e^{-\lambda y}. \tag{4.17}$$

So, the higher the peak, the less likely it is noise. Taking the logarithm of the odds, the score of a matched peak pair is

$$\text{score}(m, m') = \log \frac{\mathbb{P}(\text{peak } m' \text{ is signal at } m)}{\mathbb{P}(\text{peak } m' \text{ is noise})} = \log \operatorname{erfc}\left(\frac{|m - m'|}{\sqrt{2}\,\sigma_{\text{mass}}}\right) + \lambda y.$$

Since the noise probability appears in the denominator, we simply add the peak intensity, multiplied by the constant λ representing the noise in the spectrum, to the score. Finally, we can use prior probabilities, computing the odds that any peak is not noise: We add a constant b, being the logarithm of these prior odds, to the score of each matched peak. The score of the complete spectrum is the sum of scores over all matched peaks; for additional and missing peaks, see Sec. 4.6.

4.10 Likelihood score for high mass accuracy, both spectra with intensities

Sections 4.7 and 4.8 allow us to derive a mathematically elegant score in case the matching is perfect (that is, there are no missing or additional peaks). For a matched pair (i, j), we add up the logarithms of the mass deviation probability and of the intensity likelihood (4.16), omitting its constant factor:

$$\text{score}(i, j) := \log \operatorname{erfc}\left(\frac{\left|m_i - m'_j\right|}{\sqrt{2}\,\sigma_{\text{mass}}}\right) - \frac{(h_i - h'_j)^2}{2\left(\sigma_{\text{abs}}^2 + (h'_j)^2 \sigma_{\text{rel}}^2\right)}.$$

For peaks missing in one of the spectra, we do not have an experimental mass deviation between the two peaks. But we cannot simply ignore this mass deviation in our scoring: Otherwise, it may be preferable to mark two matching peaks with low intensity both as "missing", to avoid the penalty for matching the masses — all our probabilities are smaller than one and, hence, the logarithm is always negative. For such a peak, we therefore assume a mass deviation whose two-sided tail probability is $1 - \hat{p}$, that is, we add $\log(1 - \hat{p})$ to the score. The parameter $\hat{p} \in (0, 1)$ is not much more than a workaround for this issue, but

at least it has a clear statistical interpretation: We assume that the non-observed peak has mass deviation that, according to our statistical model, is as extreme as for $\hat{p}$ of all observed mass deviations, see Fig. 4.2. So, $\hat{p} = 0.5$ corresponds to the quantiles of the normal distribution (25 % on the left and 25 % on the right end of the normal distribution), whereas $\hat{p} = 0.8$ assumes that mass deviation is as for 10 % on the left and 10 % on the right end of the distribution. Admittedly, $\hat{p}$ is an annoying parameter (not a nuisance parameter). Since small peaks have larger mass deviations than large peaks, as the exact mass of the peak is harder to determine for the peak picking software, it is more reasonable to set $\hat{p} = 0.8$ than $\hat{p} = 0.5$; but you can also do some statistics on your data at hand to establish an even better parameter choice.

4.11 Smarter ranking: Beyond the one-size-fits-all score

Currently, there appears to be a misconception in the field of computational mass spectrometry, which says that the ultimate way to search in a database is to compare the query spectrum with candidate reference spectra that have intensities. (Such reference spectra can either be measured from standards, or possibly simulated.) As soon as we have such reference spectra in our database, we are done: We simply use a dot product (Sec. 4.1 or 4.4) to find the best candidate.

But this is not at all what database searching is about! Let us take a step back: What we want is a method that ranks the candidates such that the correct candidate is found on the top rank as often as possible; more precisely, that for all queries, correct candidates are ranked "as well as possible". How we get this ranking is not important for the task: Usually, we rank the candidates using some score of the query against each candidate, but not even that is required. (As an example, take a look at the "learning to rank" literature in machine learning.) Having reference spectra with intensities is definitely a plus; but there is no reason to believe that a direct comparison of spectra via, say, the dot product will result in the best-possible ranking.

I suggest that the way to build a better score, is to have a very close look at the application. Let us consider peptide identification via tandem mass spectrometry (CID, to be precise): If both the b and y ion are present in the measured spectrum, this is a better indication than observing a b or a y ion; even if the intensity of the single peak is higher than the summed intensities of both b and y ion. (This is one explanation why using the square root of intensities might actually improve results.) Another example is that y ions tend to appear in consecutive series, so a "ladder" of five y ions should be scored better than five y ions distributed with gaps across the peptide sequence. Hypothetically,[3] certain peak pairs may "get lost" in experimental spectra as twins: If we looked at hundreds of experimental tandem mass spectra of the same peptide, we would always find both peaks or none. In this case, the presence of both peaks should be rewarded, the absence should not be penalized, whereas the presence of only one of the peaks in the query spectrum should be substantially penalized. None of that can be taken into account when simply comparing the query spectrum to a reference *spectrum*.

Even if you are directly comparing query and reference to compute a score for ranking, certain applications require special treatment: Analyzing isotope patterns (Chapter 8), we may "trust" peak intensities to an extent that is much higher than for any other application: From a chemical standpoint, all peaks belong to exactly the same molecule. Also, masses (or, more precisely, mass differences) can be "trusted" much more for this application. See Sec. 8.3 for details. On the other hand, small peaks might easily get lost in LC-MS peak picking, and should not be penalized — or, actively searched for in the raw data.

An example against the one-size-fits-all scorings is Percolator by Käll, Canterbury, Weston, Noble and MacCoss [167]. Using a "bag of hits" from a complete LC-MS run with thousands of peptides to be identified, it extracts features from these hits and compares them to those of decoy hits, see Chapter 6. It then uses a linear support vector machine [167] or a nonlinear neural network [333] to re-score the peptide-spectrum matches, maximizing the number of matches accepted at a given false discovery rate. This substantially improves the number of hits, but comes at the risk of overfitting: That is, we might get

[3] I have never observed such twins in the data, but I assume that they may exist. If not, Bugger!

results that *look like* what we expect, instead of the true identifications. See Check [54] for a prominent example of overfitting in proteomics.

A second example is PepNovo by Frank and Pevzner [116]. Here, different peaks corresponding to one cleavage of the peptide (abcxyz ions, water losses, ammonia losses and so on, see Sec. 2.6) are no longer scored individually. The idea is as follows: If we see an intense peak which we could explain as an a ion for the current peptide candidate, but the corresponding b ion and y ion are absent from the experimental spectrum, does it really make sense to give the candidate a positive score for that? PepNovo models the dependencies between ion types using a Bayesian network.

Another example of a "non-standard" score is MS-GF, which is described in Sec. 5.4. MS-GF uses a "regular" additive score first to compare the measured spectrum with a peptide sequence; but on top of that, it computes the p-value and E-value that this score or higher is reached by chance. The resulting E-value score is calibrated (Sec. 5.5), which is desirable for significance estimation via decoy databases.

Finally, for metabolites, many computational methods never consider reference spectra for the structure candidates; instead, structural information is directly inferred from the mass spectra. These computational methods perform substantially better than those that try to simulate and compare reference spectra [317]. Examples include CSI:FingerID [93] for general metabolites and the Lipid Data Analyzer [146] for lipids. See Chapter 10 for details.

Nevertheless, there exist applications where comparing against reference spectra with intensities is "the way to go", be they measured (spectral libraries) or predicted — at least, nobody had a promising alternative idea so far. Undoubtedly the most important such application is DIA and SWATH: So far, we have assumed that a fragmentation spectrum contains fragments from a single compound structure. MS instruments measure such spectra by first scanning through the MS1 spectrum, then iteratively selecting the k most intense peaks there for fragmentation, where k depends on the user settings. For each peak, a small m/z window is opened, and molecules with this m/z are then fragmented. Even there, it is not unlikely that more than one compound structure is selected for fragmentation; but such *chimeric spectra* are considered bad quality, or, in the most common approach, researchers simply ignore that some of the spectra are chimeric. After recording the k individual fragmentation spectra, the process is repeated. This is called Data-Dependent Acquisition (DDA).

But today, more and more people are using Data-Independent Acquisition[4] (DIA) such as SWATH from the lab of Ruedi Aebersold [125]. Here, we do not select one m/z at a time; instead, we fragment all molecules that have m/z in a window of, say, 25 units. By cycling this window through the m/z range (say, m/z 400 to 1200), we have recorded fragmentation spectra of all molecules in the mass range. After recording 32 spectra, the process is repeated. Unfortunately, this means that chimeric spectra are now the rule rather than the exception; and many fragmentation spectra will contain fragment peaks from, say, 10+ compounds. Here, searching in a spectral library is a powerful and simple way to disentangle the chimeric fragmentation spectrum, in particular if the reference spectra have peak intensities.

4.12 Hypothesis-driven recalibrating

So far, we have assumed that mass spectrometry can measure the mass-to-charge ratio of an ion; unfortunately, this is not true. A mass spectrometer can only measure a derived physical property such as time of flight, or that the molecule will pass through a quadrupole filter on a stable trajectory for some voltages. Physical properties are transformed into mass-to-charge using a *calibration function*. The function itself is determined by the physics of the instrument; but its coefficients are determined using a separately measured calibration spectrum that contains *molecules of known mass*. Doing so, we can determine which, say, time-of-flight has to be mapped to which mass-to-charge. Unfortunately, subtle changes of instrument parameters can cause mass inaccuracies in the sample mass spectra: As an example, consider time-of-flight mass spectrometry. Ionized molecules are accelerated in an electric field with constant

[4] Hilarious acronym…

force F, then drift for some time through the field-free flight tube before they hit the detector. We measure the time t that a molecule needs to get from the source to the detector. To simplify things, assume a singly charged ion, where acceleration solely depends on its mass, $a = \frac{F}{m}$. Furthermore, let us ignore the time of acceleration in our calculations. If the ion is accelerated over distance d, it reaches velocity $v = \sqrt{2ad}$, and its time-of-flight over the drift distance l is $t = l/\sqrt{2ad}$. Solving for m yields $m = \frac{2Fd}{l^2} \cdot t^2$. Accounting for a time offset, the acceleration phase we ignored, and numerous other details of the physical reality we ignored (such as the initial velocity of ions), one uses a calibration polynomial $m = c_0 + c_1 t + c_2 t^2$ with three coefficients. Coefficients are estimated using calibrants of known mass. Now, assume that the temperature in the lab changes by, say, one degree Celsius. This means that the flight tube changes its length (shrinks or expands). Unless the instrument is again calibrated before the next measurement, this implies that the coefficients of our calibration function are no longer correct and, hence, mass-to-charge values determined by that function differ from the true mass-to-charge of the analyzed ions! This can easily result in deviations of more than 20 ppm for measured mass-to-charge values.

We now want to improve the mass accuracy of peaks in a tandem mass spectrum. The concept of recalibration is to use hypothetical knowledge of the investigated sample, to recalibrate the measured spectrum. For example, peptides or glycans can only generate fragments of certain masses. A "global calibration" is possible if we find peak masses in the measured spectrum that can be assigned to exactly one theoretical mass. But such peaks are hard or impossible to find particularly in the high mass region, see Sec. 8.6 for peptides and Sec. 8.5 for metabolites. In contrast, such "global recalibration" is possible for glycans as the weighted alphabet is small, see Chapters 3, 11 and Goldberg *et al.* [127].

We now concentrate on *hypothesis-driven recalibration* which can be applied for any type of molecules, but which we will explain in the context of peptide tandem mass spectrometry. In a nutshell, this works as follows: We are given a measured spectrum, and we want to search for a best hit in a peptide database. For that, we iterate over the peptides; for each peptide s, we simulate a spectrum. But before we now score the measured spectrum against the simulated one, we insert a recalibration step: At that point in time, we are investigating the hypothesis that the measured spectrum corresponds to peptide s. We also know that the calibration of the measured spectrum is defective, as the calibration function has been calculated from a different spectrum. So, let us use our hypothesis (peptide s) to correct the calibration of the measured spectrum! The ideal peak masses of the simulated spectrum obviously have better quality than the measured peak masses. So, let us find a set of peak pairs from the simulated and the measured spectrum that are "sufficiently close"; then, use the masses of the simulated spectrum, to correct *all* masses in the measured spectrum. Note the difference to the usual recalibration using compounds of known mass: Here, the "compound of known mass" is our hypothesis, and we recalibrate the measured spectrum anew for every hypothesis. After scoring the candidate, the recalibration is discarded, and we proceed with the next candidate. This is possible whenever we have a list of candidates to score: in a database search, for the peptide or glycan candidates of the two-step approach (Sec. 2.8), or for candidate molecular formulas; for example, SIRIUS recalibrates for every candidate molecular formula [28].

To make this approach work in practice, recalibration has to be robust, as we do not want to recalibrate using wrongly assigned peak pairs, and fast, as we do it for every peptide in the database. Regarding *robustness*, we note that there is a multitude of peptides with almost identical mass; for example, K and Q, AD and W, or SV and W, see Sec. 8.6. Also, we should not recalibrate on two or three peak pairs; we need many more pairs to detect outliers. Furthermore, the recalibration algorithm has to be *fast*, since recalibration must be performed for every simulated spectrum that shows at least some similarity to the measured mass spectrum.

In the following, we construct a *linear* mapping $f(x) = ax + b$ from the peak masses of the measured spectrum to the masses of the simulated spectrum, and use it to correct all peak masses in the measured spectrum. Restricting ourselves to linear mappings allows for very fast methods for this task.

We formalize the calibration task as a *linear one-dimensional point set matching problem*: Given two sets of real values, i.e., one-dimensional point sets $A, B \subseteq \mathbb{R}$, find a linear function $f : \mathbb{R} \to \mathbb{R}$ such that $\left|E_f\right|$ is maximum, where E_f is the edge set of a bipartite graph on A, B such that $\{a, b\} \in E_f$ if and only

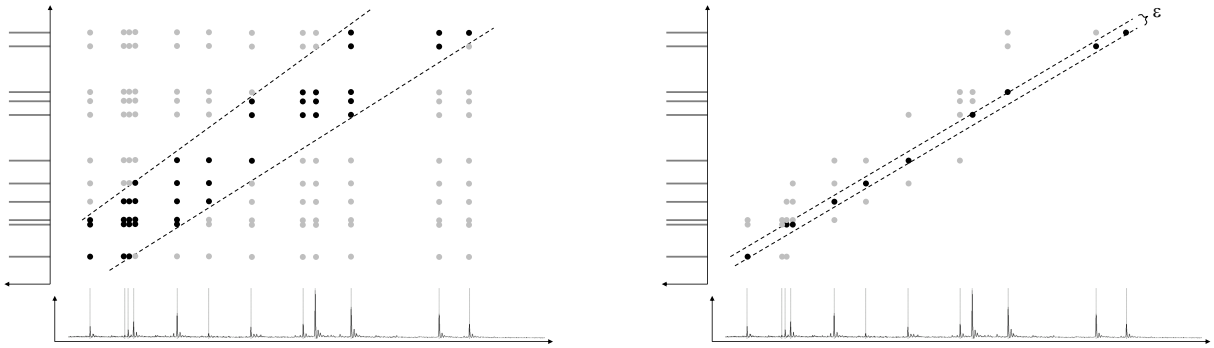


Figure 4.3: On the geometric interpretation of the problem. Left: selecting the set S of point pairs from the two spectra that are reasonable for recalibration. Right: Finding two parallel lines with fixed vertical distance w stabbing a maximum number of points.

if $|f(a) - b| \leq \varepsilon$. Note that some $a \in A$ can be mapped into ε-distance of several $b \in B$ and vice versa. In fact, in most instances, there is a degenerate optimum solution mapping all points of A into ε-distance from one point of B. In our application such degenerate cases can be avoided by restricting the search space: The measurement technique gives some absolute limits for the maximum scale and translation values. Within that range of transformations, degenerate solutions are rare. In our application, A is the set of measured peak masses, and B is the set of masses of the simulated spectrum. Parameter ε is the largest mass error that we accept after recalibration; pairs with larger error are outliers and do not contribute. A reasonable value for ε depends on the instrument and other conditions.

We next use a geometrical interpretation of the problem to find an efficient algorithm for mass spectra recalibration. In the *Maximum Line-Pair Stabbing* (MLS) problem, we are given a set of N points in the plane, and want to find a pair of parallel lines at vertical distance w such that the number of input points in the strip between the two lines (the points *stabbed* by the line pair) is maximized. In the following, we present an algorithm that solves MLS in time $O(N^2 \log N)$ and space $O(N)$.

How do we transform the problem of mass spectra recalibration to an instance of MLS? Recall that A, B denote the sets of mass values. We define a set of points in the plane $S := \{(a, b) : a \in A, b \in B\}$ and try to find a line pair that stabs a maximum number of points in S, see Fig. 4.3. As we accept deviations $|f(a) - b| \leq \varepsilon$ in both directions, the two lines must have vertical distance $w = 2\varepsilon$. By this, we construct a point set matching that allows many-to-many mappings of A to B. To exclude degenerate cases, we assume that scale $s \in [s_0, s_1]$ and translation $t \in [t_0, t_1]$ are bounded by some intervals. Then, we can restrict our set of points in the plane,

$$S := \big\{(a, b) : a \in A, b \in B, \text{ and } b \in [s_0 a + t_0 - \varepsilon, s_1 a + t_1 + \varepsilon]\big\}. \tag{4.18}$$

Nonetheless, the solution will in general not define a one-to-one mapping between A and B: For distinct $a, a' \in A$ and $b, b' \in B$ with $|a - a'| \ll \varepsilon$ and $|b - b'| \ll \varepsilon$, the optimal line-pair may stab all four points (a, b), (a, b'), (a', b), and (a', b').

See Algorithm 4.1 for a solution to the line pair stabbing problem. The algorithm is not very complicated, and does not use any involved concepts from computer science. But where does this algorithm come from, and how does it work? Now, this *is* complicated, and brings us into the realm of computational geometry; see de Berg *et al.* [76] for an excellent introduction to this topic.

Our solution is based on the *duality transform* of a set of points in the plane introduced by Brown [45]. The *dual* of a point $p = (p_x, p_y)$ in the plane is the line $p^* : y = p_x x - p_y$, while the dual of a line $q : y = q_x x + q_y$ is the point $q^* = (q_x, -q_y)$. The vertical distance between a point p and a line q equals

```
1:  function RECALIBRATE(set S = {(x_1, y_1), ..., (x_n, y_n)} of mass pairs)
2:      score* ← −∞
3:      for i = 1, ..., n do                                   ▷ the first point/line
4:          Q ← empty list of events; c ← 0
5:          for j = 1, ..., n with j ≠ i do                    ▷ the second point/line
6:              δ_x := x_i − x_j; sign := sign(δ_x)
7:              if δ_x ≠ 0 then
8:                  a := (y_i − y_j)/δ_x; append (a, sign) to Q
9:                  a' := (y_i + w − y_j)/δ_x; append (a', −sign) to Q
10:             else if 0 ≤ y_j − y_i ≤ w then
11:                 c ← c + 1                                  ▷ stabbed for every slope
12:             end if
13:         end for
14:         Sort list Q with respect to first entry, and entries with sign = +1 before those with sign = −1 in
        case of ties
15:         score ← c
16:         for all (a, sign) in the sorted list Q do          ▷ find the maximum
17:             score ← score + sign
18:             if score > score* then
19:                 score* ← score; a* ← a; and b* ← x_i · a − y_i
20:             end if
21:         end for
22:     end for
23:     If required, process vertical lines with infinite slope
24:     Extract subset S' ⊆ S of those (x_i, y_i) satisfying 0 ≤ y_i − a* · x_i + b* ≤ w
25:     Use Ordinary Least Squares to derive â, b̂ from S'
26:     return (â, b̂)
27: end function
```

Algorithm 4.1: Recalibrate a measured spectrum to a reference spectrum. Input is a set of mass pairs belonging to peaks that might match. Parameter w is the vertical width of the strip, that is, twice the maximal mass error we accept after recalibration. Line 23 is not needed for the recalibration problem, as recalibration slope a is close to one and far from infinity.

the vertical distance between the line p^* and the point q^*. Furthermore, the dual transform maintains the above/below relationship between a point and a line. See e.g. [76, Chapter 8.2] for more details.

We now describe our solution to the MLS problem. We are given a distance w and a set $S \subseteq \mathbb{R}^2$ of points in the plane. In the following, the distance between two parallel lines is not the Euclidean distance, but their vertical distance. Let us ignore vertical line pairs that can be handled separately. Consider two parallel lines $q : y = q_x x + q_y$ and $q' : y = q_x x + q_y + w$. The lines parallel to q between q and q', including q and q', are mapped to the line segment $\{q_x\} \times [-q_y - w, -q_y]$ in the dual. Finding a line-pair that stabs a maximum number of points in S, is equivalent to finding a line segment $\{x\} \times [-y - w, -y]$ such that the number of intersected lines in S^* is maximum, over all choices of x and y. Note that the optimal line segment intersects the lines in S^* in some order, so there exists a first and a last line stabbed.

The idea of the algorithm is simple: We fix one line p^*, compute the intersections of every other line q^* with p^* and with the line parallel to p^* at distance w, and sweep over these intersections from left to right. In detail, we iterate over all lines $p^* \in S^*$, and assume that p^* is the first line stabbed. Note that one of the lines has to be stabbed first and, since we are iterating over all lines, this is no restriction. Every other line $q^* : y = q_x x - q_y$ partitions p^* into a constant number of ranges as follows: Only in the range between the intersection of p^* and q^*, and the intersection of q^* with the line parallel to p^* and at distance w, can this line contribute to a line segment that stabs p^* first (see Fig. 4.4). Projection to the x-axis leads to the

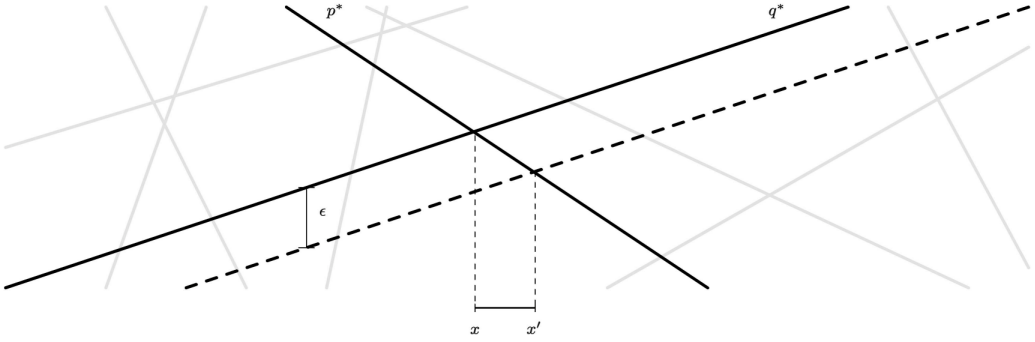


Figure 4.4: Maximum line-pair stabbing algorithm: finding the interval where q^* can contribute to a segment starting in p^*.

interval bounded by points $x = (p_y - q_y)/(p_x - q_x)$ and $x' = (p_y - q_y + w)/(p_x - q_x)$. Attaching +1 or −1 to the endpoints depending on whether the endpoint is start or end of a range and then sorting these endpoints, one can scan through the endpoints keeping a counter of how many ranges are active. The optimal choice for $(x, -y)$ corresponds to the overall largest count: The line l with slope x through point p, together with the parallel line l' at vertical distance w above it, stabs a maximum number of points in S.

The above algorithm solves the point set matching problem in time $O(|S|^2 \log |S|)$. For unrestricted scale and translation, S contains all $|A| \cdot |B|$ pairs, resulting in time $O\big((|A|\,|B|)^2 \log(|A|\,|B|)\big)$.

4.13 Historical notes and further reading

Even before the dot product and as early as 1971, other scores were used for searching spectral libraries of small molecules, see Stein and Scott [336]. There exist several approaches from the metabolomics community that try to improve on the dot product for spectral library searching [188, 259, 267, 336], usually by weighting masses and intensities: For example, Stein and Scott [336] found that for the dot product, taking the square root of intensities and weighting with the square or cube of the mass works best.

Similarly, there exist numerous approaches for searching in peptide databases, some of which have become commercial; I mention only a few: MOWSE (MOlecular Weight SEarch) by Pappin *et al.* [264] was targeted at the identification of proteins using peptide mass fingerprints. Its successor MASCOT by Perkins *et al.* [270] is commercial; it tries to convert the MOWSE score into a p-value. SEQUEST by Eng *et al.* [103] is one of the first tools for searching peptide fragmentation spectra in databases, and is now commercial. In early versions, the software rounded all masses to integers and "smeared" the barcode reference spectrum, assigning half the intensity to the masses ±1 next to each predicted peak. It then computed the dot product with the measured spectrum, and subtracted the mean dot product over mass shifts between −75 and +75 (cross-correlation, computed via the fast Fourier transform). The smearing resembles the Gaussian peak shapes of Sec. 4.4, which may explain why SEQUEST performed very well in practice; on the downside, rounding all masses to integers throws away the mass accuracy of the measurement. X!Tandem by Craig and Beavis [66] is an open source tool available from Global Proteome Machine Organization.

Back in 1992, Owens [262] suggested using correlation of mass spectra as a score. Probability product kernels were introduced by Jebara, Kondor and Howard [163]. In fact, these kernels are much more general than what I have presented here; the complete Sec. 4.4 would fit into a few lines of Jebara *et al.* [163]. But for the purpose of comparing and scoring mass spectra, the kernel as presented in Sec. 4.4 is sufficient.

An additive scoring for rewarding observed peaks and penalizing unobserved peaks was possibly first suggested by Dančík *et al.* [70], together with a stochastic justification of this principle ("premium for present ions, penalty for missing ions"). In fact, I have the feeling that the more elaborate scorings were developed as part of *de novo* analysis of data, such as peptide *de novo* sequencing. This may be explained by the fact that *de novo* sequencing is many orders of magnitude harder than database searching: Simply

compare the number of considered candidates. So, an elaborate score was simply not needed in database searching.

Using absolute peak intensity differences for scoring was proposed by Pluskal *et al.* [273], using relative peak intensities by Böcker *et al.* [31]. The combined model was introduced by Dührkop *et al.* [90]. In all three cases, the authors analyzed isotope patterns.

Using peak intensities to score the peaks in a spectrum, as explained in Sec. 4.9, has been proposed many times in the literature [227], but this is usually done without giving any (stochastic) justification. Goldberg *et al.* [127] suggested using $\exp(a_0 + a_1 x + a_2 x^2)$ to model intensities of noise peaks; but it appears that their estimated a_2 is *very* close to zero (see Fig. 4, right in [127]), so that we are back to an exponential distribution. Also, their model has the unappealing property that noise peaks with negative intensity have probabilities larger than zero. The fact that the distribution is truncated for low intensities, can be attributed to thresholding in the peak picking algorithm and has no "deeper" meaning.

Zubarev and Mann [379] propose to recalibrate using confidently identified peptides as internal calibrants, so that the remaining mass errors are normally distributed around zero. The paper also contains some details on mass accuracy needed to identify peptides and proteins from their monoisotopic mass.

Simulated mass spectra usually have no intensities or "rule-of-thumb" intensities. An exception is again tandem MS of peptides, where millions of training examples are available and spectra can be predicted by machine learning. This was proposed as early as 2004 [102], see also Degroeve and Martens [79]; deep learning was first used for peak intensity prediction by Zhou, Zeng, Chi, Luo, Liu, Zhan, He, and Zhang [378] in 2017, see also Gessulat *et al.* [124] and Tiwary *et al.* [348]. Funnily enough, we also worked on predicting peak intensities of peptides back in 2008, see Timm, Scherbart, Böcker, Kohlbacher and Nattkemper [346]. For small molecules, peak intensities are predicted by CFM-ID [1, 357] and, more recently, by deep learning methods such as MassFormer [374] and ICEBERG [131].

4.14 Exercises

4.1 You want to compare two unit mass spectra, but only one of them has peak intensities. What is the dot product actually measuring?

4.2 Show how to combine two sorted lists of size m and n into one in time $O(m+n)$.

4.3 Write an algorithm to simulate the tandem MS spectrum of a peptide if only b and y ions are present, and ions have a single proton.

4.4 Describe an algorithm that computes the probability product kernel of two ideal spectra m_i, h_i and m'_j, h'_j in "basically linear" time, assuming that the peak lists are sorted by mass.

★ 4.5 Give a formal description of what "basically linear" means for the previous algorithm.

4.6 Given two peak lists $\mathcal{M} = \{150, 175, 220, 310, 470\}$ and $\mathcal{M}' = \{150, 190, 250, 315, 485\}$. Calculate the peak counting score for $\varepsilon = 5$ and for $\varepsilon = 15$.

4.7 Let $\text{score}(i,j) = 2 - \frac{1}{5}\left|m_i - m'_j\right|$ and $\text{score}(i,\text{-}) = \text{score}(\text{-},j) = -1$. Calculate an optimal alignment of these two peak lists.

★ 4.8 Assume a monotone mass scoring function, and gap penalties that do not depend on the peak. Show that the candidate partners of each peak m_i form a contiguous block in $\mathcal{M}'$, and that these blocks move monotonically with i.

4.9 Why is it reasonable, from a mathematical perspective, to assume that for 10 ppm mass accuracy, the difference between two peak masses has at most 14 ppm? Hint: If a random variable X is normally distributed, then $-X$ is so, too. Under what condition is it reasonable, from an MS perspective, to assume that this mass difference is still 10 ppm or better?

4.10 Write an algorithm to simulate the tandem MS spectrum of a peptide as thoroughly as possible.

4.11 How much does the length of a steel tube change if the temperature rises by one degree Celsius? How much does that change the mass we determine from the formula $m = \frac{2Fd}{l^2} \cdot t^2$ for a flight tube of length l?

4.12 In the classification above, the *many-to-one* matching, where one peak in the reference may be matched against multiple peaks in the query, but each peak in the query is matched against at most one peak in the reference, is hardly ever relevant in applications. Explain why.

★ 4.13 Build a scoring or ranking for peptide tandem mass spectra which is not simply comparing peaks in the query and the reference.

5 Significance: p-values and E-values

> *"The grand assertion is that you must see the world through probability and that probability is the only guide you need."* (Dennis Lindley)

> *"What do you think of my new poem?" — "I once read that given infinite time, a thousand monkeys with typewriters would eventually write the entire works of Shakespeare." — "But what about MY poem?" — "Three monkeys, ten minutes."* (Dilbert & Dogbert)

The content of the following two chapters is a little different from the rest of this textbook, as it deals with statistics and stochastics of mass spectrometry analysis rather than combinatorics and algorithmics. This overview will be short and vastly incomplete: In fact, a complete textbook can be written about the statistical analysis of peptide and protein MS, which has many similarities but also some unique features compared to transcriptomics. The reasons to include these chapters are twofold: First, I want to argue that what I present here is the minimum information you need to do sensible work in algorithmic mass spectrometry. Second, computational MS only makes sense in light of statistics: Computational MS is about "real data" and, as such, full of inaccuracies, errors, misclassifications, and spurious signals. Usually, the best way to deal with these problems is statistics. In Chapter 4, I have indicated how to modify some optimization algorithms so that results have "statistical meaning". Finally, the ideas described in the following two chapters can be reused in many other areas of computational mass spectrometry and science.

In the previous chapter, we have seen how to match a measured spectrum to a reference spectrum. Again, we will focus on the task of identifying a peptide using MS/MS data, but solely for the sake of readability: The methods presented here can in principle be used for other applications, as long as the requirements are met. We search our measured spectrum against a database of reference peptide sequences, and we accept the reference spectrum and, hence, the reference peptide with the highest score as being the correct answer. We call the pair "measured spectrum" plus "best candidate peptide sequence" a *hit* in the database. Hits are often referred to as *peptide-spectrum matches* (PSM), but I do not see the necessity for yet another acronym (YAA).

Unfortunately, the best-scoring peptide is not necessarily the correct answer. This can be due to two different phenomena: First, the correct answer is part of the database we search in, but a different peptide received a better score; see the Charlie Chaplin example in Chapter 6. Second, the correct answer is not part of the database: There are many potential causes for this, such as an incomplete protein database for the organism we are looking at, protein from a different organism in the sample, impurities through sample preparation such as keratin (the key structural material making up the outer layer of human skin), data from a cyclic (non-ribosomal) peptide, data from a metabolite or glycan, or tandem mass spectra that do not contain any real biomolecules but only "noise". (I was very surprised how often MS instruments record these "empty spectra", where you cannot spot any peak at the precursor mass in the MS1.) For such spectra, our method will also find a best hit in the database. In both cases, we will call such hits *spurious* or *bogus*.

How can we differentiate between true hits and bogus hits? Is a score of 120 a good score and, hence, a true hit? We can compare it to other scores but maybe, all of our hits are bogus, and all scores are bad scores. The most reasonable way to deal with this dilemma, is to estimate the statistical *significance* of a hit. We will consider two concepts to estimate significance, namely p-values and E-values (this chapter); and false discovery rates and q-values (next chapter).

5.1 p-values and E-values

We are given a single query tandem mass spectrum; we have selected the best candidate (reference spectrum) from a list of N candidates according to maximum score. The principle underlying p-value computation is that the score T of the hit (score between query and best-scoring candidate) also allows us to differentiate between true and bogus hits. We define the *p-value* of a hit as the probability that we observe a score of at least T for a random reference spectrum. Accordingly, our null model is that the reference spectrum is random; our null hypothesis is that the observed or a better score is reached by chance. If the p-value is very small, we can reject the null hypothesis, and infer that the hit is "significant". Nothing else must be interpreted into this test; for example, if our hit has p-value p then $1-p$ is *not* the probability that the hit is correct, see Sec. 6.6 below. Note that a small p-value means high significance, and a large p-value means low significance.

Clearly, we need a statistical model (null model) of random reference spectra that we want to consider: Formally, let Ω be the sample space, $\mathcal{M} \in \Omega$ be the (elementary) events, and $p(\mathcal{M})$ be the probability of reference spectrum $\mathcal{M}$. Then, the p-value is the sum of $p(\mathcal{M})$ where the score of the query spectrum vs. reference spectrum $\mathcal{M}$ is at least T. It is understood that p-value estimates depend on the chosen null model; it should also be understood that, by choosing a bad null model, we can come up with p-value estimates which are useless. See below for two examples, one bad and one reasonable null model.

Next, the *E-value* is the expected number of events that pass the score threshold T if we repeat this experiment N times; we do so as we have N candidates in our candidate list, and any one of them may pass the score threshold by chance. It is easy to see that the E-value is $N \cdot p$ for p-value p, compare with coin flipping and the binomial distribution.

In theory, there exist *three approaches to assign p-values* to a hit with score T: Here, we have considered randomizing the targets, that is, the reference spectra or peptide sequences. But instead, we can also randomize the query, that is, the sample mass spectra; third, we can randomize both the query and the targets. I am not aware that someone has used these alternative routes in shotgun proteomics, as it is non-trivial to randomize query mass spectra. If you are not searching peptides but other macromolecules such as glycans, the situation is similar to shotgun proteomics: It is rather straightforward how to randomize the candidates, and how to transform a candidate structure into a reference spectrum. In metabolomics, though, it is highly non-trivial to randomize metabolite structures and to transform them into mass spectra; but we can randomize query spectra [311]. For comparison, in local sequence alignment, randomizing both query and targets is the "industry standard" Karlin-Altschul statistics [174] first used in BLAST [4], see also below. See Sec. 5.6 for issues of the three approaches.

What is a reasonable background model, that is, a reasonable set Ω of reference spectra to choose from? In "the old days" of computational mass spectrometry, some people proposed using mass spectra with random peak masses as Ω: Simply draw peak masses at random, for example, uniformly distributed in the interval $[0, M]$ where M is the precursor mass of the measured spectrum. Here, the number of peaks may be chosen as the average number of peaks of a reference spectra database. Unfortunately, this is a very bad background model. Consider shotgun proteomics: Due to the experimental setup, most of the measured mass spectra will actually correspond to *some* peptide, even though the peptide sequence might not be recorded in the database. In Chapter 2 we have seen that peptide fragmentation spectra have a particular structure, which is clearly not the case if we generate spectra using random peaks. Even if our database hit is bogus, it might share some peaks with the measured spectrum, possibly because a few amino acids agree with the measured peptide we are searching for; see also Sec. 6.4. In contrast, randomizing peak masses will make it unlikely to find a single matching peak pair. In total, we will grossly overestimate the significance, that is, compute a p-value which is much too small. This stays true if peak masses are drawn with respect to some empirical distribution computed from, say, a reference database: Peak masses in a peptide fragmentation spectrum are highly correlated, and independently drawing peaks neglects these dependencies.

But since we search in a database of peptide sequences and simulate the reference spectra anyway, it is clearly a smarter choice to limit ourselves to reference spectra that correspond to some peptide sequence. We have to go even further: We only want to consider those peptides that have the correct precursor mass, as this is the filter we use for our reference database search, too. Therefore, the sample space Ω contains all peptide strings with the correct precursor mass. We may choose different probabilities $p(s)$ for different random peptides $s \in \Omega$: For example, we can take into account relative abundances of amino acids in our database so that peptides which use many "uncommon" amino acids are considered with smaller probability.

5.2 A naïve approach for estimating p-values

Assume that our measured spectrum $\mathcal{M}'$ (the data) was scored highest against reference spectrum $\mathcal{M}^*$ from the database, and reached score T (the true score). To randomize the reference, we have to sample a large number of random reference objects, score each random object against the data, and count the number of times this score is greater than or equal to the true score. In detail, let Ω be the space of reference spectra. Randomly choose a reference spectrum $\mathcal{M} \in \Omega$ according to probability $p(\mathcal{M})$; we say that we "sample" from Ω. Score the reference spectrum against the measured spectrum $\mathcal{M}'$, computing the score $\text{score}(\mathcal{M}, \mathcal{M}')$. Repeat this "a reasonable number of times". Count the number of random reference spectra $\mathcal{M}$ with $\text{score}(\mathcal{M}, \mathcal{M}') \geq T$. Divide by the number of repetitions, to compute an empirical p-value. In fact, a better p-value estimate is $(k+1)/(n+1)$ if there are k random spectra above the threshold, and we are considering a total of n random spectra [72].[1]

The problem of the naïve approach stems from the fact that we have to repeat "a reasonable number of times". We are usually interested in very small p-values; for most applications, there is a huge difference between p-value 10^{-5} and 10^{-10}. This is because we want to compute E-values or have to correct for multiple testing (Sec. 5.5); for $N = 10000$ a p-value of 10^{-4} means that we expect that one random reference spectrum will pass the score T just by chance. This is not really convincing evidence that the hit is correct, is it? But to reach a non-zero p-value of 10^{-10} we have to repeat at least 10 000 000 000 times. This will result in prohibitive running times: Even if we could score 10^9 random spectra against the sample spectrum per second, calculating a single p-value would still require ten seconds; and this now has to be done for every hit.

But let us ignore this problem for a moment, and assume that we have enough time to do the sampling. Then, the question is: What is Ω, and how do we sample from it? We have argued above that for shotgun proteomics, a reasonable choice for the sample space Ω is the set of all strings over Σ with mass M. For the moment, let us assume we have no further background information, and all strings $s \in \Omega$ have the same probability $p(s) = 1/|\Omega|$. The algorithmic question is: How can we sample *uniformly* from Ω? The important word here is "uniformly": That is, each string with mass M is drawn with the same probability. A naïve approach that just adds random characters to the string and restarts every time we exceed mass M, does not even sample uniformly — a string s is generated with probability $|\Sigma|^{-|s|}$, so short strings are preferred — and it takes many tries to generate a single string. A less naïve approach that checks whether a decomposition for mass m exists (see Chapter 3), unfortunately violates uniformity (Exercise 5.1).

Luckily, we have already solved this problem, possibly without noticing it: Recall from (3.3) that the number of strings $C'[m]$ of mass m can be computed by the recurrence $C'[m] = \sum_{a \in \Sigma} C'[m-a]$, where $C'[m] = 0$ for all $m < 0$. Remember that $a \in \Sigma$ denotes both the character and its mass. Now, we generate the string from right to left, starting with the empty string $s \leftarrow \epsilon$ and $m \leftarrow M$. We randomly draw a letter $a \in \Sigma$ where each a has probability $C'[m-a]/C'[m]$; it is clear from the recurrence that these values add up to one. Character a is appended to s on the left side, $s \leftarrow as$, and we set $m \leftarrow m - a$. We repeat until

[1] Using $(k+1)/(n+1)$ instead of k/n is again some kind of statistical magic; unless you can explain off the cuff why we have to divide by $n-1$ when we compute the sample variance (Bessel's correction), it might be too much detail. I have to tell you that I cannot. For $(k+1)/(n+1)$, the reason is simpler, though: Under the null hypothesis, the true score is just one more random score; this estimate is never zero, and it does not overstate significance. Claude helped!

$m = 0$. It is straightforward to check that this algorithm uniformly draws a string with mass M, as each string is chosen with probability $1/C'[M]$.

I suggested generating the string s from right to left, because this is what backtracing through the dynamic programming table would do; and this is also what we do here, only in a probabilistic fashion. In practice, it does not matter whether you generate the string from right to left or from left to right, because $s = s_1 \dots s_l$ and the inverse string $s^{-1} = s_l \dots s_1$ have the same mass. (Note that we say "inverse string" rather than "reverse string": In bioinformatics, the latter is easily confused with the reverse complement of a DNA sequence.) So, you can also go from left to right, which is usually easier to write as code.

You might argue that you do not want to sample uniformly from Ω, and that certain peptide strings should be sampled with higher or lower probability. But now that we can sample uniformly, we can also incorporate such deviations. For example, we can easily incorporate that amino acids are not chosen uniformly but instead have individual probabilities π_a for $a \in \Sigma$: Let S be an infinite string where each character a has probability π_a to be appended, $\sum_a \pi_a = 1$. The probability $p[m]$ that a prefix of S has mass m, can be computed as

$$p[m] = \sum_{a \in \Sigma} \pi_a \cdot p[m - a] \tag{5.1}$$

where $p[0] = 1$ and $p[m] = 0$ for $m < 0$. To sample a string s of mass M, we again start with an empty string and iteratively add characters, where character a is selected with probability $\pi_a \cdot p[m - a]/p[m]$.

5.3 Parametric distributions

Let us take a short detour: The first time bioinformatics students come across p-values and E-values, is probably in the context of BLAST (Basic Local Alignment Search Tool). Here, scientists faced the same problem: Computing a local alignment between two sequences, what does a score of T "mean"? Karlin and Altschul showed that scores of two random sequences can be modeled via a random walk, and *proved* that scores of the *optimal alignment* follow an extreme value (Gumbel) distribution. Hence, we can calculate the p-value for score T using the cumulative distribution function of the extreme value distribution,

$$\mathbb{P}(X \geq T) = 1 - \exp\left(-K \cdot mn \cdot e^{-\lambda T}\right)$$

where X is a random variable for the optimal alignment score of two random sequences. Estimating exact parameters K and λ of the distribution is part of the proof; these correspond to parameters $\mu = \ln(Kmn)/\lambda$ and $\beta = 1/\lambda$ for the standard form of the extreme value distribution with cumulative distribution function

$$\exp\left(-e^{-(x-\mu)/\beta}\right).$$

The extreme value distribution is one example of a parametric distribution; other examples are the normal distribution, the exponential distribution, the Gamma distribution, the log-normal distribution, or the Pareto distribution. Being able to prove that scores follow a particular distribution, is undoubtedly the best we can come up with; clearly, our proof is then valid only for one particular score from Chapter 4. Usually, this is not possible for us. But even for Karlin-Altschul statistics, proofs only hold for local sequence alignments *without insertions and deletions*. It was "empirically established" that the score distribution was sufficiently close to an extreme value distribution for local sequence alignments with InDels, using extensive simulations with *many millions of queries and target databases*. The parameters of the extreme value distribution were also determined by these simulations, for several combinations of score matrix and affine gap penalties.

How can you "empirically establish" that scores follow a particular parametric distribution? To this end, we use 100 (better, 1000) query spectra; in the following, we assume that one query is fixed. We then plot the distribution of scores the query reaches against 1000 (better 100 000) random candidates. This is basically the same procedure as for the naïve estimation of p-values in Sec. 5.2, except that we record all

Distribution	parameters	mean	variance	skewness
normal	$\mu \in \mathbb{R}$, $\sigma^2 > 0$	μ	σ^2	0
exponential	$\lambda > 0$	$1/\lambda$	$1/\lambda^2$	2
Gamma	$k > 0$, $\theta > 0$	$k\theta$	$k\theta^2$	$2/\sqrt{k}$
Gumbel	$\mu \in \mathbb{R}$, $\beta > 0$	$\mu + 0.577216\,\beta$	$1.644934\,\beta^2$	1.139547

Table 5.1: Mean, variance, and skewness of some parametric distributions. The Gumbel distribution is more precisely known as "Generalized Extreme Value distribution Type-I", and is often referred to as "extreme value distribution" in bioinformatics.

scores, and not only how many scores passed the threshold. From a statistical viewpoint, the score is a *random variable* X; the 1000 to 100 000 scores of the fixed query against random candidates are a *sample* for this random variable. You can visualize the distribution of scores using a bar plot or — usually more instructive — a kernel density: Kernel density estimation is a non-parametric estimation of the probability density function of a random variable. We then hopefully see that these empirical distributions look similar to the same parametric distribution, for each query.

In reality, a score distribution may look like a normal distribution for small scores, and like a log-normal distribution for high scores. What now? The answer is obvious and somewhat brutal: Nobody cares about high p-values. If you estimate a p-value of 0.6 when in truth, it should have been 0.9 — well, I honestly could not care less. Both p-values tell us that the score is not at all significant. What we want to avoid is that the p-value estimate is 0.1 for a true p-value of 10^{-10}. In theory, you could and should exclusively look at the right end (say, top 1 %) of the empirical score distributions, and ignore what is going on for the other 99 %. In practice, doing so will make it hard to identify "the correct" parametric distribution. But if a parametric distribution fits the complete empirical score distribution for some instances, whereas we observe a good fit for the top 30 % of scores for other instances, then this is as good as it gets. Unfortunately, this can complicate parameter estimation for the parametric distribution, see below.

Let us assume ideal conditions: For each query, the resulting kernel density does in fact look like, say, the probability density function of a Gamma distribution. So, this part is fixed; whatever query we are given, we assume that scores of candidates follow again a Gamma distribution. But obviously, for different queries, this is not the same Gamma distribution: We observe that mean, variance and skewness are different, for each query. How can we adapt the parametric distribution to the query spectrum?

Clearly, what we have to do is to choose appropriate parameters of the parametric distribution. Each parametric distribution comes with its own parameters, see Table 5.1. An instructive way to choose the parameters, is via the moments: For each parametric distribution, we know the moments such as mean, variance and skewness, see again Table 5.1. Moments can be infinite for certain parametric distributions: For example, the Pareto distribution with parameters $x_\mathrm{m} > 0$, $\alpha > 0$ has mean $(\alpha x_\mathrm{m})/(\alpha - 1)$ for $\alpha > 1$ but mean ∞ for $\alpha \leq 1$; similarly, it has variance ∞ for $\alpha \leq 2$ and skewness ∞ for $\alpha \leq 3$.

On the other hand, moments can be estimated from the data for our random variable X: Let $x_1, \ldots, x_n$ be the sample, that is, the scores of a query spectrum versus n random targets. Computing the sample scores is again the same procedure as for the naïve estimation of p-values in Sec. 5.2, and we again record all scores. But this time, it is sufficient that we sample, say $n = 100$ or even fewer random candidates! It is well known that unbiased estimators $\overline{x}$ for sample mean and S^2 for sample variance are

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad S^2 = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \overline{x})^2 .$$

Higher moments such as skewness can be estimated in a similar fashion. Now, we can determine the parameters of the parametric distribution using the moments of the sample (that is, the observed scores), using the known moments of parametric distributions from Table 5.1. For example, consider the Gamma

distribution: We may choose parameters k, θ as $k = \overline{x}^2/S^2$ and $\theta = S^2/\overline{x}$, then $\mu = k\theta = \overline{x}$ and $\sigma^2 = k\theta^2 = S^2$ as desired.

Using estimated moments to derive the parameters is an easy-to-follow argument but *not advisable in practice*. Three issues should be immediately clear: First, for the Gamma distribution, how can we be sure that value $S^2/\overline{x}$ is positive, as is required for this distribution? Second, we have implicitly chosen $2/\sqrt{k} = 2/\sqrt{\overline{x}^2/S^2}$ as the skewness of the Gamma distribution, completely ignoring the skewness of the actual sample. Third, we learned that for certain parametric distributions, some moments are not even finite. But the most important point is that there is *no statistical justification* why we should choose the parameters of the distribution in this way!

Instead, you should use maximum likelihood to estimate the parameters of the distribution. This is easy for us because luckily, someone smarter than me has already established maximum likelihood estimators for pretty much any parametric distribution you can think of; you may look them up in some statistics textbook or in Wikipedia. For example, the maximum likelihood estimator $(\widehat{x}_\mathrm{m}, \widehat{\alpha})$ for the parameters of the Pareto distribution is

$$\widehat{x}_\mathrm{m} = \min_{i=1,\dots,n} x_i \quad \text{and} \quad \widehat{\alpha} = \frac{n}{\sum_{i=1}^n \ln(x_i/\widehat{x}_\mathrm{m})}$$

where $x_1, \dots, x_n$ are the samples (observed scores). For the normal distribution, we estimate parameters $(\hat{\mu}, \hat{\sigma}^2)$ as

$$\hat{\mu} = \overline{x} = \frac{1}{n}\sum_{i=1}^n x_i \quad \text{and} \quad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \overline{x})^2 .$$

Note that the maximum likelihood estimator for sample variance, $\hat{\sigma}^2$, is not Bessel-corrected.

Unfortunately, estimating the parameters in this way does not necessarily work in case only high scores follow the assumed parametric distribution. For example, low scores may behave chaotically for combinatorial reasons, whereas the top 10 % of scores nicely follow a parametric distribution. But we can fit the distribution to the high scores only: We choose a threshold u — in our example, the 90 % quantile of the scores — and maximize the likelihood of the scores above u, conditional on being above u, using density $f(x)/(1-F(u))$ where f and F are density and cumulative distribution function of the parametric distribution. If k_u of our n samples lie above u, then the p-value of a score $x > u$ is

Claude helped!

$$\text{p-value} = \frac{k_u}{n} \cdot \frac{1-F(x)}{1-F(u)}.$$

Extreme value theory even tells us which distribution to use: Scores above a sufficiently high threshold approximately follow a generalized Pareto distribution [60, 272]. Choosing the threshold u is, unfortunately, again a trade-off.

What did we gain by replacing the naïve method of p-value estimation by the detour via a parametric distribution? On the one hand, we only have to generate a relatively small number of score samples, such as $n = 100$, for any given query. This is usually enough to reach high-quality estimates for mean and variance, but also to estimate the parameters of the distribution. On the other hand, we can now estimate p-values via the known cumulative distribution function of the parametric distribution: The *cumulative distribution function* at x is the probability that a real-valued random variable will take a value at most x. Hence, the p-value of x is simply one minus the cumulative distribution function at x. The cumulative distribution function of a parametric distribution usually has some simple closed form: For example, assuming that scores follow a Pareto distribution with parameters x_m, α, then the cumulative distribution function is $1 - (x_\mathrm{m}/x)^\alpha$, and the p-value of score x is simply $(x_\mathrm{m}/x)^\alpha$. (Recall that the most prominent exception is the normal distribution, as no closed form for the definite integral of the probability density function is known; in practice, we can compute the p-value using the complementary error function provided by numerical libraries, see Sec. 4.7.) Finally, estimated p-values can get *arbitrarily small* when the score of the target candidate x gets large; in particular, this is independent of the number of samples we used to estimate parameters. But do not confuse "can be computed" with "is accurate": For a p-value

Claude helped!

of 10^{-10}, we extrapolate far beyond our samples, and with $n = 100$ samples, the estimate may easily be off by orders of magnitude, even if the parametric distribution is the correct one. In practice, this is usually irrelevant: A hit with p-value 10^{-12} is significant, and so is a hit with p-value 10^{-9}; and the order of hits does not change.

Sometimes, people report *z-scores* instead of p-values. This is simply the number of standard deviations some value is above or below the sample mean, and can be estimated as $z = (x - \bar{x})/S$ where $\bar{x}$ is the mean of the sample $x_1, \ldots, x_n$, and S is its standard deviation. Reporting z-scores means that we are agnostic of the underlying distribution. Unfortunately, when this is done in practice, some readers (and maybe even writers) will *implicitly assume* that values are normally distributed; in this case, a z-score of +5 is highly significant, not to mention +7. But this is a false conclusion: Values may also follow a Pareto or log-normal distribution, and for these distributions, being seven standard deviations above the mean is not that surprising, see Exercise 5.4. See also Sec. 5.6 below.

5.4 Exact computations using dynamic programming

We now turn to a method for exact computation of p-values. This method is called MS-GF, where GF stands for "generating functions". Generating functions allow us to do involved mathematical tricks such as multiplication, division, or taking the derivative of the functions, which usually are infinite series. None of this is actually required here, so we will use a much simpler mathematical formalism based on random variables and the convolution of distributions. Computations from this section will be reused in Chapter 7, so it is worth reading this section even if you are not interested in the application.

Assume that you are given an ideal die. You will model this stochastically using a discrete random variable $X : \Omega \to \{1, \ldots, 6\}$ where Ω denotes the sample space (everything that might happen). The probability that a particular value $x \in \{1, \ldots, 6\}$ is reached, is $\mathbb{P}(X = x) = \frac{1}{6}$, and zero everywhere else. Assume that we have a second die with random variable Y, and we want to model the sum of these two dice. One can easily see that the sum of the dice, $X + Y$, has distribution

$$\mathbb{P}(X + Y = x) = \sum_{y=1,\ldots,6} \mathbb{P}(X = x - y) \cdot \mathbb{P}(Y = y). \tag{5.2}$$

This can be generalized beyond dice: For two random variables $X, Y : \Omega \to \mathbb{N}$ we have

$$\mathbb{P}(X + Y = x) = \sum_{y=0,\ldots,x} \mathbb{P}(X = x - y) \cdot \mathbb{P}(Y = y). \tag{5.3}$$

and if both random variables have finite support (that is, only a finite set of numbers has probability strictly greater than zero) then this is actually a finite sum.

It is now simple to actually compute these probabilities for $X + Y$: Let $P_X[0 \ldots x_{\max}]$ be the array with $P_X[x] = \mathbb{P}(X = x)$ and $\sum_{x=0,\ldots,x_{\max}} P_X[x] = 1$, and $P_Y[0 \ldots y_{\max}]$ analogously. Then, we can compute $P_{Y+X}[0 \ldots x_{\max} + y_{\max}]$ as

$$P_{Y+X}[x] \leftarrow \sum_{y=0,\ldots,y_{\max}} P_X[x - y] \cdot P_Y[y] \tag{5.4}$$

where we assume $P_X[x] = 0$ for $x < 0$ and $x > x_{\max}$.

This is all the mathematics that we need in this section. We again over-simplify our problem slightly, to improve readability. To this end, assume that all masses are integers, that our peptide only generates prefix masses, and that we use the peak counting score. Let the peak scoring function $f(m)$ indicate whether a (prefix) peak is present ($f(m) = 1$) or absent ($f(m) = 0$) in the query spectrum at mass m: The score of a string s is simply

$$\sum_{s' \text{ is prefix of } s} f\big(\mu(s')\big).$$

This score is not limited to strings of mass M; in particular, we can compute it for prefixes of such strings.

Recall recurrence (5.1) from Sec. 5.2, which allows us to sample strings with mass M where character a is drawn with probability π_a. There, we computed $p[m]$ as the probability that a prefix of an infinite random string has mass m. Here, we define $P[i,m]$ as the joint probability that a prefix s with mass m of the random infinite string explains exactly i peaks, $f(s) = i$. Notably, $\sum_{i=0}^{\infty} P[i,m] = p[m]$: Every prefix has to generate *some* score. We can compute the $P[i,m]$ using the recurrence

$$P[i,m] = \sum_{a \in \Sigma} \pi_a P\big[i - f(m), m - a\big] \tag{5.5}$$

where we initialize $P[f(0),0] = 1$ and $P[i,0] = 0$ for $i \neq f(0)$, and implicitly assume $P[i,m] = 0$ for $i < 0$ or $m < 0$. The correctness of the recurrence is relatively easy to see if you have understood recurrence (5.1). Recall that a is both the character and its mass. Clearly, we do not have to consider an infinite number of scores; in fact, we have to consider only $i = 0, \dots, T-1$ where T is the (peak counting) score of the database search hit. Now, the joint probability that a prefix of the random string with mass M has score at least T, is $p[M] - \sum_{i=0}^{T-1} P[i,M]$. But we condition our calculations by considering only those prefixes that have exactly this mass; hence,

$$\text{p-value} = \frac{1}{p[M]} \cdot \Big(p[M] - \sum\nolimits_{i=0}^{T-1} P[i,M]\Big). \tag{5.6}$$

is the p-value we are searching for: The probability that a random string of mass M will have score at least T.

Our first assumption that masses are integers is not restrictive in practice, see the discussion in Chapter 3 and Sec. 8.8 below, as well as many other places in this textbook. Our third assumption about using the peak counting score is slightly harder to get around: Clearly, we want to score peaks differently, depending on peak intensity and mass deviation. Accordingly, there is some score $f'(m) \in \mathbb{R}$ that a prefix with mass m will add to the total score of the candidate. Here, we have to discretize scores as $f(m) \in \{0, 1, \dots, f_{\max}\}$, as we have done for masses. Again, we can limit computations in the recurrence to scores that are at most the score of the database hit we are evaluating.

But what about the fact that we are only considering prefix masses? It is straightforward that we can take into account other ions besides the b ion in the prefix mass score $f(m)$. But we cannot avoid the peak double counting issue: If a string has a prefix *and* a suffix of mass m, then the corresponding peaks in the spectrum will be counted twice, as both $f(m)$ and $f(M-m)$ will be part of the sum that scores the string. We have spent all of Chapter 2 on how to avoid this issue; and now we claim that it is not of particular importance?

Turns out that we can indeed ignore the double counting issue. (See Sec. 2.10 for a heuristic approach on how to avoid double counting for peptide *de novo* sequencing.) Here, we are searching in a peptide database, and we cannot freely choose a string that scores the most intense peaks in the experimental spectrum (the prefix masses m with close-to-maximum $f(m)$) twice. Therefore, it is *unlikely* that the database hit is scoring any relevant peaks twice. But what about recurrence (5.5) and Eq. (5.6)? There, we are considering *all* strings with precursor mass M; and this includes all strings that do contain prefix and suffix of mass m for high-scoring $f(m)$. The answer is the same: Such strings are *unlikely* and, hence, scoring them "slightly awkward" will not change our p-value computations substantially.

In the title of this section, we have promised an exact method, and that's what it is — as long as our restricting assumptions are satisfied. In practice, you can score the peptides for database searching with a continuous scoring where neither masses nor peak scores have been discretized, and you might also avoid peak double counting. But you can nevertheless use the above calculations — where masses and scores were discretized — to get a highly accurate "approximation" (in the informal sense, see Sec. 15.7) of the true p-value.

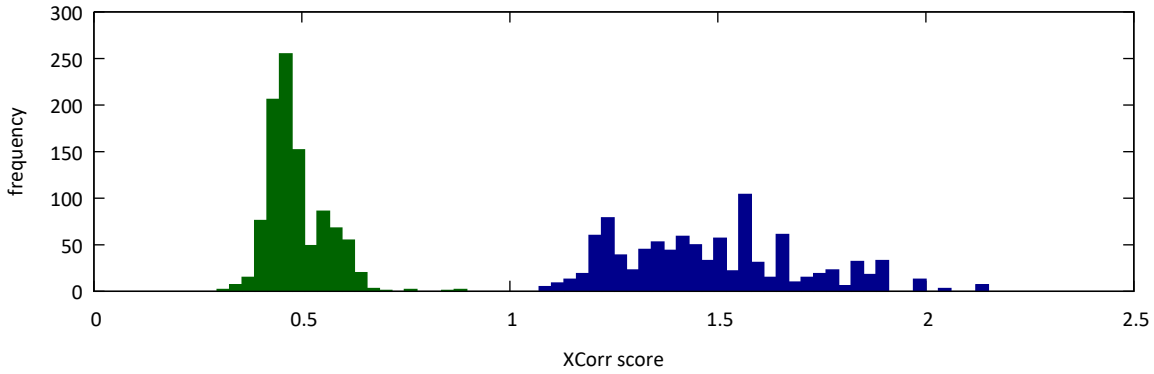


Figure 5.1: Score distributions for two experimental tandem mass spectra against 1000 random peptide sequences each (shuffled decoys, see Sec. 6.4 and Exercise 5.8). We observe that all scores of the first spectrum (green) are smaller than all scores of the second spectrum (blue). Choosing a score threshold of 1, we will regard all random hits of the first spectrum as "not significant" and all random hits of the second spectrum as "significant". Figure redrawn from Keich and Noble [178], spectra from a yeast dataset.

5.5 Calibrated scores and p-value correction

Remember the question we started off with: Is a score of 120 a good score and, hence, is the corresponding hit correct? We were unable to decide this based on the score, as it depends on the query spectrum and the precursor mass: In the extreme case, one query spectrum may receive a higher score against any candidate, than another query spectrum against any candidate! See Fig. 5.1 for a nasty example. This is why we introduced p-values and E-values.

It turns out that this is exactly what we will do for false discovery estimation in the next chapter: There, we will order hits by score, assuming that the high-scoring hits are more likely correct than the low-scoring ones. But that means that we can indeed "trust" the score in the sense that a score of 120 indeed means a good hit, under all circumstances! So, it makes much more sense to order hits by p-value or E-value, to decide whether they are correct or incorrect.

But rethinking p-values, we see that they are not apt for this purpose, either: It makes a huge difference if a hit (the highest-scoring candidate) was chosen from a set of 10 or 10000 candidates! Throwing three dice, a roll of 18 is surprising; rolling them 10000 times and reporting the maximum, we would be surprised if the maximum is less than 18. Hence, we must not sort by p-value. In contrast, E-values have been corrected for "multiple testing"; hence, we can compare E-values from different queries, and rank queries according to E-value. We call this a *calibrated score* [178].

Luckily, the false discovery rate estimation procedure described in Sec. 6.3 is very robust against the use of uncalibrated scores, and estimates are still of high quality if we use our everyday score such as the dot product. Nevertheless, it is somewhat upsetting to use the hit score to decide if a hit is correct, when we already know that *the score is not apt for this purpose*. It turns out that even for the robust method from Sec. 6.3, the number of hits at a given false discovery rate increases, see Keich and Noble [178]. You can decide for yourself.

It must be understood that a score being calibrated is not a yes/no question; rather, there are numerous shades of gray on how well a score is calibrated between different queries and corresponding hits.

So, the E-value can serve as a calibrated score; but it is not a probability, and can take values beyond 1. When we want to further work with probabilities, for one reason or another, then we rather want to *correct* the p-value for the multiple testing (taking the maximum score over N candidates).

We can correct the p-value for multiple testing as follows: The p-value that the hit (the best-scoring candidate) reaches score at least T is $1-(1-p)^N$ if there are N candidates and p is the p-value of the

highest-scoring pair "query plus candidate" for score T. The formula is easy to understand: Not a single candidate must have reached the score threshold by chance. Here, we assume that the scores of the N candidates are independent, as in our null model; for a real database with overlapping and homologous peptides, this holds only approximately. Unfortunately, if you use this formula directly, you will experience some surprises, see Sec. 15.3: The formula is numerically unstable. Luckily, any reasonable math library contains a function that computes $f(x) := \log(1+x)$ with high precision when $|x|$ is small; similarly, there exists a designated function to compute $g(x) := \exp(x) - 1$ for $|x|$ small. With these functions, you can now compute

$$1 - (1-p)^N = 1 - \exp\big(N \log(1-p)\big) = -g\big(N \cdot f(-p)\big). \tag{5.7}$$

Function names for f, g are, for example, "log1p" and "expm1".

When $N \cdot p \ll 1$ is much smaller than one, then

$$1 - (1-p)^N = 1 - \sum_{i=0}^{N} \tbinom{N}{i}(-p)^i 1^{N-i} = 1 - 1 + Np - \tbinom{N}{2}p^2 + \tbinom{N}{3}p^3 \pm \cdots \approx N \cdot p,$$

because $N \cdot p \ll 1$ implies

$$N \cdot p \gg \tbinom{N}{2}p^2 = N(N-1)/2 \cdot p^2 \gg \tbinom{N}{3}p^3 = N(N-1)(N-2)/6 \cdot p^3 \gg \ldots.$$

So, $N \cdot p$ can serve as an approximation for the corrected p-value. But you should only use this approximation when you are *sure* that $N \cdot p$ is much smaller than one: All you gain from this approximation is that you can compute it without executing f, g in (5.7).

5.6 Issues of p-value estimation

Unfortunately, there are several issues with p-value computation; some of them were mentioned previously. We will discuss them here, so that you are warned when interpreting p-values. I stress that *any reasonable* p-value computation is much preferred to not computing p-values; the discussion below is rather meant to "keep you alert" and prevent you from trusting p-values blindly. There are too many papers where authors wrongly pray to the gods of p-values [160]. We first concentrate on randomizing the targets, as done in shotgun proteomics, and cover randomizing of the query later.

We noted above that (estimated) p-values depend on our null model; different null models will result in different p-value estimates. In particular, a bad null model will result in "bad" p-values, meaning that these p-values do not carry any useful information.

Using a parametric distribution for p-value estimation, we face the threat that the distribution is not the correct choice for the dataset we want to analyze right now. Maybe, the chosen distribution was fine for the datasets we looked at during methods development; but how can we be sure this still holds true for the current dataset? Only a formal proof that scores follow a particular distribution, as for local sequence alignments and Karlin-Altschul statistics, can protect us from unpleasant surprises. Visual inspection to choose the "correct" parametric distribution can easily be misleading; choosing the wrong distribution can again have a devastating effect on our p-value estimates. For example, if we use a short-tailed distribution (normal, exponential) when the true distribution is heavy-tailed (Pareto, log-normal), we will substantially overestimate significance: Our p-value estimates will be much — potentially, many orders of magnitude — smaller than the true p-values, see Exercise 5.4.

Next, the true distribution may be bimodal, something not covered in the parametric models; the second mode might be hard to spot because it is much smaller than the first, major mode. Unfortunately, the second (tiny) mode of the score distribution might contribute more to the p-value than the first (large) mode if the score we are considering is close to the second mode. Ignoring the second mode will again result in inflated significance estimates, see Exercise 5.5. You can easily make things worse by considering a third mode which is even smaller than the first two, and so on. From my experience, I can say that

multimodal score distributions are the rule rather than the exception; see also Fig. 5.1. Unfortunately, not all queries will result in a multimodal distribution. Note that exact estimation of p-values (Sec. 5.4) does not suffer from the last two problems.

I noted that p-values must not be misinterpreted as "probabilities that a hit is incorrect". But p-values also cannot provide information about the "quality" of a spectrum: It is possible that an identification from a low-quality spectrum reaches a small p-value, whereas an identification from a high-quality spectrum may only reach a large p-value. This can happen independently of the correctness of the hit. Assume we have a perfect copy of some reference spectrum as our query; assume further that the hit reaches a very small p-value (say, 10^{-30}). Now, add noise to the query spectrum, but make sure that all added peaks cannot be explained by *any* peptide sequence. For example, we may add peaks at mass m + 0.25 Da for integer and reasonably small m, say, below 100; or, even better, at mass $1.00048924 \cdot m$ + 0.25 Da, where the factor accounts for the mass defect of peptides (see averagine in Chapter 8). Such peaks can only be explained by fragments that carry four charges, which is practically impossible for all applications we consider in this textbook. For all scoring functions we have considered in Chapter 4, this will change the score of any reference; but this change will usually be via a monotonic function, meaning that the *order of scores* is not changed. In particular, the target hit score is ranked at the same position among the scores against random references. Hence, we will estimate exactly the same p-value for the disturbed query spectrum as for the perfect match (again, say 10^{-30}); and this is still the case if the noise peaks are responsible for 99 % of the total intensity in the query spectrum. Would you trust this identification, in particular at this level of significance?

But wait, there is more! What about the alternative approaches to assign p-values? Randomizing the query also comes with certain issues: First, this approach has repeatedly been criticized for ignoring the actual query in p-value estimation. For example, Karlin-Altschul statistics [174] will treat two queries TWTWTW and SSSSSS that reach the same hit score exactly the same, because they also have the same length. This is despite the fact that, according to amino acid frequencies, one peptide is very rare whereas the other is rather common. This is acceptable for sequence comparison because we are using a scoring matrix, and treat amino acids differently when computing the score. Modern versions of BLAST partially address this via composition-based statistics.

But how can we generate a random query spectrum that somehow represents the peculiarities of our query spectrum, given that our scores for comparing mass spectra are "agnostic" to the peculiarities of the underlying biochemical question and experimental setup? What are the important characteristics of our query spectrum that must be mirrored in the random spectra so that p-value estimates are reasonable? Can we formally describe the sample space we use in our computations, or is the sample space defined only implicitly via the algorithm to generate random spectra? If this is the case, we have no possibility to check via theoretical considerations that resulting p-values are "probably useless".

5.7 Further reading and other approaches

The problem of wrongly assuming a score distribution to be normal by reporting the "number of standard deviations above the mean," was already pointed out in 1994 by Waterman and Vingron [362] for pairwise sequence alignments. The BLAST paper by Altschul, Gish, Miller, Myers and Lipman [4] is one of the most-cited papers of all time, with more than 75000 citations in March 2019. The successor paper [3], introducing gapped BLAST and PSI-BLAST, was cited a mere 70000 times.

Sampling random strings of a fixed precursor mass was proposed by Lu and Chen [219]. The method for exact computation of p-values for peptides, MS-GF, was proposed by Kim, Gupta and Pevzner [187]. It is also described in Chapter 11 ("Was *T rex* Just a Big Chicken?") in Compeau and Pevzner [61].

Numerous parametric distributions have been proposed for the scores introduced in the respective papers: For example, PepProbe [304] uses a hypergeometric distribution, X!Tandem [110] a Gumbel distribution, OMSSA [122] a Poisson distribution, and Crux [265] a Weibull distribution.

In proteomics, to the best of my knowledge, all approaches for "randomizing a tandem mass spectrum" still require that we know the peptide sequence that generated the spectrum. These approaches first assign labels ("a-ion" etc.) to the peaks in the spectrum, then disturb the spectrum using the known labels. Therefore, we cannot use these approaches to randomize the query spectrum, as this is exactly the information that we do not have. In metabolomics, randomizing queries can be achieved using PASSATUTTO by Scheubert, Hufsky, Petras, Wang, Nothias, Dührkop, Bandeira, Dorrestein, and Böcker [311], as long as you have a batch of queries; the paper applies it to the reference spectra, but it works for query spectra just as well. Unfortunately, this method cannot be applied to peptides, as it ignores the linear nature of peptides. Furthermore, it faces the issues of randomizing the query mentioned in Sec. 5.6.

Keich and Noble [178] called calibrated scores "well-calibrated scores", to make clear that this is not a binary thing. I am lazy, so I saved one word. See there and in the next chapter on how calibrated scores help in false discovery rate estimation. Keich and Noble [178] also found that the E-value estimates from Sec. 5.5, while being much better than "regular scores", are still not perfectly calibrated. This might be due to the fact that the exact estimation does not take into account peptide length, see Sec. 6.4. It is straightforward to take peptide length into account for the DP in Sec. 5.4, but it will add a factor l to the running time for target peptide length l, see Exercise 5.8.

Numerous articles have been written that warn against "bad statistics" and "bad interpretation" thereof [160]. Be reminded that this must not be used as an excuse for *not using statistics*.

5.8 Exercises

5.1 The "not-so-naïve" method to sample a string of mass M is to build the string iteratively from right to left, and to consider for the remaining mass m if there exists a string (or compomer) of mass m. Give an example where this approach *does not* uniformly sample a string of mass M.

5.2 Compute $1-(1-p)^N$ and $N \cdot p$ for $p = 10^{-3}$ and $N = 1000$.

5.3 Compute $1-(1-p)^N$ and $N \cdot p$ for $p = 10^{-10}$ and $N = 1000$ using double precision and without the designated functions; compare to the exact value.

5.4 Assume that scores are distributed according to a Pareto distribution with parameters $x_m = 1$ and $\alpha = 3$. Calculate mean, variance and standard deviation of the distribution. What is the probability of being 10 standard deviations above the mean (z-score 10)? What would be the estimated p-value if we had wrongly assumed that the score distribution is normal? Repeat calculations for 20 standard deviations above the mean (z-score 20).

5.5 Assume scores against random candidates have a mixture distribution: In 99.9999 % of the cases, the score is drawn from the normal distribution $\mathcal{N}(0,1)$; and in 0.0001 % of the cases, the score is drawn from $\mathcal{N}(8,1)$. What is the p-value of score 10? What would be our p-value estimate if we had wrongly determined that the score distribution is simply $\mathcal{N}(0,1)$? Why is it rather likely that we would have wrongly estimated the latter score distribution in practice?

5.6 Modify computations from Sec. 5.4 so that precursor masses between m_1 and m_2 are taken into account.

5.7 Speed up computations from Sec. 5.4 by algorithm engineering: Notice that not all masses m have to be computed, because both the prefix mass and the suffix mass must have a decomposition. You may assume that you have computed an appropriate data structure (such as the residue table from Sec. 3.6, see Exercise 3.18) during preprocessing. How much does this trick help?

★ 5.8 Modify the recurrence from Sec. 5.4 so that only peptides of a particular length l (alternatively, peptides with length between l and u) are considered. Do some algorithm engineering so that you

do not have to iterate over all masses. (There should be a total of *four* bounds for the masses you consider, two for the lower bound and two for the upper bound.)

★ 5.9 Consider some weighted alphabet Σ. What is the number $N[M]$ of strings with mass M that have a proper prefix and suffix of the same mass? Recall that $C'[M]$ is the number of strings of mass M. Warning: A prefix-suffix pair can have a sub-prefix-suffix pair of identical mass, so $\sum_m C'[m]C'[M-2m]C'[m]$ (see Exercise 3.12) is unfortunately not the solution...

⋆ 5.10 Implement the calculations from the previous exercise, and verify for some examples that $N[M] \sim C'[M]$ for $M \to \infty$.

⋆ 5.11 Let Σ be the alphabet of amino acid residues with integer masses, where leucine and isoleucine are treated as a single character. Plot the probability that a string of mass M has prefix and suffix of the same mass.

⋆ 5.12 Repeat the above, but modify your computations to take into account that b ion and y ion differ in mass by 18 Dalton.

5.13 Assume that peptide masses are distributed according to a Poisson distribution with mean 1200. Using the numbers from the previous exercise, calculate the probability that a peptide has prefix and suffix of the same mass.

6 Significance: Decoy Databases and False Discovery Rates

> *"[Back in 1915] Charlie Chaplin look-alike contest became a popular form of entertainment. At these events, contestants would compete to see who could best imitate the 'tramp' persona championed by Chaplin. [...] According to entertainment folklore, Chaplin himself once entered and lost one of these contests. [...] Charlie Chaplin did not come in second or third, he did not even make the finals."* (Mario Cruz)

Statistical significance estimation via p-values is definitely better than nothing; but in a way, this is answering a question that we did not ask. In particular, the p-value is *not* the probability that the hit is incorrect, and cannot be easily transformed into this probability. On the contrary, a rather natural question is as follows: Recall that in a shotgun proteomics experiment, we do not search for a single query spectrum inside the peptide database. Instead, we have thousands of query spectra that we want to identify, see Sec. 2.9. Our question is: From the set of hits (that is, the putatively identified peptides), how many are correct and how many are bogus? Assuming that the hit score does indeed allow us to separate correct from bogus hits, can I choose a subset of hits so that the fraction of bogus hits is reasonably small? This is exactly what false discovery rates are about; false discovery rate *estimation* deals with the problem that we are not omniscient (bummer). Next, q-values allow us to assign significance values to the individual query spectra. Any false discovery rate or q-value is nevertheless a property of a *batch* or list of query spectra and not a single query. Finally, posterior error probabilities are indeed the probability that a particular hit is incorrect, but their estimation comes with certain pitfalls (Sec. 6.6).

As in the previous chapter, we will concentrate on peptide identification and shotgun proteomics in our presentation. Whereas FDR estimation is a general statistical procedure, the shotgun proteomics experimental setup makes it particularly easy for us to come up with sensible decoy databases. Things get considerably more complicated if we consider metabolomics, but also metaproteomics (Sec. 6.8).

6.1 False discovery rates and q-values

Recall that a *hit* is a query plus the best-scoring candidate from the database we are searching in. We are given a list (or batch) of hits, such as all putatively identified peptides from a shotgun proteomics LC-MS run. Assume that we know which of the hits are correct (true) and which are bogus (false): For k bogus hits out of n total hits, the False Discovery Rate (FDR) is simply k/n. Speaking in the context of information retrieval, the FDR equals the number of false positives (FP) divided by the sum of true positives (TP) and false positives,

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{TP} + \mathrm{FP}}. \tag{6.1}$$

We do not have to consider the complete list of hits for FDR estimation; the definition also covers the case that we return any sublist of length n containing k bogus hits. That being said, the information retrieval view becomes clearer: True positives are the correct hits in our sublist, whereas false positives are the bogus hits we put in the sublist but should not. We can calculate numerous related classification measures such as precision or recall, but the false discovery rate will do for us.

What sublists are reasonable to consider? If our assumption is correct that the hit score allows us to differentiate between correct and bogus hits, then the only reasonable sublists we have to consider are

those hits that have hit score above some score threshold, for any score threshold. In other words, we sort the list of hits by hit score, and consider the top k for any $k = 0, \ldots, n$.

If we iteratively consider the top k sublists, then going from $k-1$ to k will either increase our FDR, in case the hit at rank k is bogus, or decrease it, in case the hit at rank k is correct. In other words, returning a larger sublist may be better — that is, have smaller FDR — than staying cautious and returning the smaller list. In practice, we usually let the user decide upon an FDR threshold (s)he can accept; we then return the largest top k sublist such that the FDR is at most the user-specified threshold.

The *q-value* of a hit is the minimal FDR at which this hit is included in our output (top k list). If we have computed FDR values p_k for all hits $k = 1, \ldots, n$, then the corresponding q-values q_k are simply $q_n = p_n$ and

$$q_k = \min\{p_k, q_{k+1}\} \quad \text{for } k = n-1, n-2, \ldots, 1. \tag{6.2}$$

It is important to understand that, whereas q-values are assigned to individual hits, this is nevertheless a property of the complete list of hits: If we add query spectra to our batch, then q-values of all hits may change. In more detail, if we add correct and high-scoring hits then q-values of other hits will decrease (become more significant); if we add bogus and high-scoring hits then q-values will generally increase (become less significant). Similarly, if we change the database we search in, q-values of all hits may change, even of those where the set of candidates is identical.

Second, if you are given a single query, you cannot compute a q-value for that. This simply does not make sense. But quite obviously, you can compute a p-value for a single query.

Third, the "precision" we can reach with false discovery rates and q-values, is limited by the number of hits in our input hit list: For n hits, the smallest non-zero FDR and q-value is $1/n$, see Exercise 6.2. This is usually not so much of a problem for false discovery rates, where a threshold is provided by the user — unless the user chooses the threshold unreasonably small. But this sets apart q-values from p-values; for the latter, we can easily reach very small non-zero values such as 10^{-10}, whereas a q-value of 10^{-3} is likely the best (non-zero value) we can expect in most applications.

So far, we have assumed that we are omniscient: That is, we know for each hit whether it is correct or bogus. Such an omniscient perspective is sometimes realistic: When we are developing a computational method, we will evaluate it on reference data where we know the correct answer, see for example Sec. 6.7.

In practice, we are clearly not omniscient; hence, the rest of this chapter deals with *estimating* the FDR in a way that our estimate is "as close to the true FDR as possible". Clearly, estimating FDR implies estimating q-values. It must be understood that neither the list of hits we start with, nor the order of hits which is defined by the hit score can be influenced by our FDR estimates. This is no longer true if we use decoys to learn a better score, as Percolator does [167]; but then, we must make sure that decoys used for training are not used again for estimation. The only questions that FDR estimation can answer, are: If I can accept a certain FDR in my output, what is a reasonable hit score threshold; and looking at an individual hit, what would be the FDR threshold to have this hit in the output?

6.2 Using random number generators to estimate false discovery rates

Our first false discovery rate estimate is conceptually rather simple; this is not completely true, as pseudo-random number generation on a deterministic computer is already a highly non-trivial problem. If we have to estimate false discovery rates for n hits, we simply draw n random numbers between 0 and 1, sort them and assign them to the hits sorted by hit score. In fact, we do not even have to sort them; but our estimates will be "more accurate" if we do so. We transform FDR estimates into q-values as described above; in case we have sorted the random numbers, the q-value equals the FDR.

You may be surprised, but this is a valid FDR estimation method. It is a very, very bad one; but this is something it has in common with many other methods for FDR estimation! Numerous methods have been published where the authors *claim* (and potentially believe) that something is an FDR estimate. But the only way to show that your FDR estimates are better than random, is to *prove it empirically*, see Sec. 6.7.

For that, you need data where you know the correct answer; otherwise, it is impossible to judge how well the method performed. This has been used as an excuse why an empirical evaluation of a new method is impossible. But today, much reference data (for example, millions of peptide tandem mass spectra where the peptide sequence is known) are available for such evaluations. If no experimental reference data are available, you have to simulate such data in the most reasonable way you can think of. But without the evaluations of Sec. 6.7, an FDR estimate should be treated with the same respect as a random number.

One more point: Sometimes, methods are indeed evaluated as "being better than random". This is an extremely weak statement; it is actually very hard to construct a computational method which does not use random numbers and is nevertheless as good (or as bad) as random. If FDR estimates are better than random, they may still be completely useless; they have to be pretty accurate to be of use in practice. On the other hand, they do not have to be perfect; but we should know how much we can trust them. See again Sec. 6.7 for details.

6.3 Decoy databases and false discovery rate estimation

Decoys are hits that are always bogus; they will help us to estimate the false discovery rate, as we can compare bogus hits against decoys and bogus hits against "true candidates". We will refer to the database that we use for searching, as the *target database*. We create a second database, called the *decoy database*, which looks similar to the target database, but only contains candidates which cannot be the correct answer. We then *combine* both databases into one, and *search in the combined database*. Thus, each query results in a hit that is either from the target database or from the decoy database.

As stated above, the decoy database should look "reasonably similar" to the target database while at the same time, all hits in the decoy database should be spurious. In detail, we want the decoy database to meet the following three assumptions:

1. There is no overlap between the decoy database and the target database: That is, candidate peptides in the decoy database are not in the target database, and vice versa.

2. The correct answer (the true peptide) is *never* present in the decoy database; any hit in the decoy database is a bogus hit.

3. A bogus hit in the target database is as likely as a (bogus) hit in the decoy database.

Note that Assumption 2 is much stronger than Assumption 1: Assumption 2 not only states something about the candidates in our target database, it states something about all possible candidates, known and unknown! Assumption 1 is in the list because it intuitively makes sense; because it can be checked empirically for any given pair target plus decoy database, which is not possible for Assumption 2; and because there might be cases where the target database contains a candidate which is never the true candidate, but should still not end up in our decoy database to avoid ties. For Assumption 3, any hit in the decoy database must be bogus according to Assumption 2. It should be intuitively clear that this assumption is the crux of our method; on the one hand, it will guarantee sensible estimates (compare to Sec. 6.2) and on the other hand, it will be the hardest to establish, both empirically and by theoretical considerations.

In practice, it is not necessary that all three conditions are perfectly fulfilled: It is sufficient that the number of exceptions to these conditions is so small, that it does not interfere substantially with our estimations.

Assuming that we were successful in building a decoy database that fulfills all three assumptions; how does that help us to estimate FDR? Recall that we are given a batch of query spectra, resulting in a list of hits. Recall further that we search in the combined target plus decoy database, meaning that any hit will be in either the target or in the decoy database. Hits in the decoy database will never be reported to the user, but instead *discarded*: By Assumption 2, these hits must be bogus, so why baffle the user by

#	score	DB	FDR	q-value
37	128.1	target	0.0 %	0.0 %
124	122.8	target	0.0 %	0.0 %
12	121.2	target	0.0 %	0.0 %
950	103.1	target	0.0 %	0.0 %
730	102.3	target	0.0 %	0.0 %
217	96.4	target	0.0 %	0.0 %
918	94.8	target	0.0 %	0.0 %
333	94.3	decoy	14.3 %	
212	93.5	target	12.5 %	10.0 %
4	93.4	target	11.1 %	10.0 %
18	92.0	target	10.0 %	10.0 %
69	90.7	decoy	20.0 %	
72	89.9	target	18.2 %	16.7 %
174	87.3	target	16.7 %	16.7 %
111	86.5	decoy	25.0 %	
750	86.4	target	23.1 %	18.8 %
828	84.2	target	21.4 %	18.8 %
830	82.3	target	20.0 %	18.8 %
13	82.2	target	18.8 %	18.8 %
522	80.9	decoy	25.0 %	

Figure 6.1: Search in combined target and decoy database. Results are sorted by score. Only the top 20 hits are reported. '#' is the (completely arbitrary) number of the query. For each k, we estimate the FDR of the top k; for each target hit, we estimate its q-value.

reporting them? But we can use hits in the decoy database to estimate the number of bogus hits in the target database, as follows: Assume that among the top n hits above a certain score threshold, there are k hits in the decoy database — and, consequently, $n-k$ hits in the target database which we report to the user. Now, by Assumption 3, for each decoy hit that passed the score threshold by chance, we expect to see also one bogus hit in the target database that passes the score threshold by chance. Hence, there are also k bogus hits among the $n-k$ hits in the target database. We output the $n-k$ hits from the target database with estimated FDR $\min\{k/(n-k), 1\}$, see (6.1). (Clearly, we should not estimate false discovery rates beyond 100 %.) Note that with this estimate, the smallest non-zero FDR is $1/(n-1)$, namely for a single decoy among the top n hits.

Let us take a look at the example from Fig. 6.1. Consider the top $n = 10$ hits from this list, corresponding to score threshold 93.4: We return $n - k = 9$ target hits to the user, and we estimate that $k = 1$ hit is bogus, as we have also found one hit in the decoy database above the score threshold. So, we estimate FDR $1/9 = 11.1$ %. If we consider the top 11 hits (score threshold 92.0), the FDR estimate drops to $1/10 = 10$ %; for the top 19 hits with score threshold 82.2, we estimate an FDR of $3/16 = 18.8$ %. See Fig. 6.1 for all estimates. Recall that in practice, the user would give us an FDR threshold, and we would return the largest list such that our estimated FDR is below the threshold.

We then estimate q-values for all hits in the target database. As in Sec. 6.1, this is the smallest FDR for which a hit in the target database will be in the output; as there, we can compute the q-value using (6.2). See again Fig. 6.1 for all estimated q-values.

One peculiarity of the above approach has to be mentioned: When we compute an FDR for a list of hits, the only hit score we take into account is the worst score, among all hits! So, the two hit lists with scores [100, 100, 100, 50] and [50, 50, 50, 50] have exactly the same estimated FDR; you may assume that score 100 indicates a perfect hit. If you estimate an FDR of 50 %, then one of your perfect hits is seemingly incorrect, despite the fact that shortening the list by one hit, we probably reach an FDR of zero. You may argue that such extreme hit lists are rare, and this is why on average (or, in expectation) our FDR estimation procedure computes high-quality estimates; this behavior is nevertheless inaccurate and somewhat counterintuitive. You may further argue that one can see the difference if one looks at the q-values of individual hits; but this does not change a thing about the strange FDR estimate for the complete list. The reason why we do not take into account *all* hit scores when estimating the FDR, is that this would make things really, *really* complicated. See Sec. 6.6 for one possibility to do so.

6.4 How to create a decoy database for peptides

Having talked so much about decoy databases, the first question that comes to mind, is: How do we build one? Clearly, our goal is to satisfy the three assumptions from the previous section. We will see that it is relatively easy to satisfy the first two assumptions; our focus will therefore lie on satisfying the third assumption. Before we start, I want to stress that FDR *estimation* already implies that there is an underlying stochastic model: The accuracy of our estimates depends on the experimental data as well as the decoy database. In fact, some of the methods for building a decoy database have a stochastic component, and repeating FDR estimation will result in varying estimates. Hence, our arguments will be from a stochastic standpoint, too: We can never completely rule out that certain things happen; we can only show that they are highly improbable, and therefore do not interfere with our estimation procedure.

Since we search in a peptide sequence (structure) database, it is reasonable to also create the decoy database on the peptide sequence level. Different methods for creating a peptide decoy database have been proposed over the years. All start off from the target database either containing full protein sequences, or peptide sequences that have been digested *in silico*, see Sec. 1.6.1. We then transform each entry in the target database into an entry in the decoy database. It is not mandatory that we generate one decoy sequence for every target sequence; but in view of the third assumption, this again appears to be very reasonable.

Methods to build a peptide decoy database include:

- *Inverted proteins.* We invert all target proteins, that is, read them from right to left. Then, we do *in silico* digestion to create the peptide decoy database.
- *Inverted peptides.* We invert each target peptide, reading it from right to left; for example, PEPTIDE becomes EDITPEP.
- *Pseudo-inverted peptides.* We invert each target peptide but keep the last character in place, so $s = s_1 \dots s_{l-1} s_l$ gets $s_{l-1} \dots s_1 s_l$; for example, PEPTIDE becomes DITPEPE.
- *Shuffled peptides.* For each peptide in the target database, we generate a decoy peptide by shuffling the characters of the peptide using a uniformly drawn permutation. For example, PEPTIDE may become IDETPEP or EITDPPE.
- *Pseudo-shuffled peptides.* We shuffle each peptide but keep the last character in place. For example, PEPTIDE may become TIDEPPE or PITEDPE.
- *Random iid.* We use the target database to estimate the relative frequency of each amino acid. For each peptide of the target database, we generate a random peptide of the same length. Each character is drawn independently and with identical distribution (i.i.d.), using the amino acid frequency estimates as probabilities. For example, PEPTIDE may become MYSTERY.
- *Markov chain.* Instead of drawing the letters independently, we learn a Markov chain from the target database, and generate random peptides of identical length distribution as the target database using this Markov chain. I leave out the technical details.

Some of the above methods are deterministic, and one target database is transformed into exactly one decoy database; others are random algorithms, and will produce different decoy peptides if we run them repeatedly on the same target database.

Why do we keep the last character in place for the "pseudo"-methods? The answer is simple: For the target database, the majority of peptides end with the characters K and R (lysine and arginine), whereas these characters can rarely be found at other positions of the peptide, in which case they are followed by a P (proline). It is intuitively clear that our decoy database should also follow this restriction, to make target peptides and decoy peptides "more similar"; we will come back to this point later. It is understood that we can easily modify methods *random iid* and *Markov chain* to also take into account the peculiarities

of the last position; we leave out the technical details solely for brevity. A similar intuition is behind our requirement that target peptide and decoy peptide must have the same length, see again below for details.

What about the three assumptions from the previous section, which are mandatory so that FDR estimation via a decoy database can work? First, consider Assumption 1. In practice, this assumption is easy to check: Simply generate the decoy database, and search for overlap. But there are also some theoretical considerations telling us that this overlap can be neglected in practice: We may assume that peptides in the target database have some minimal length such as six amino acids, as shorter peptides and their fragmentation spectra are rather uninformative in practice. There are $20^6 = 6.4 \cdot 10^7$ peptides of that length — ignoring for simplicity that we cannot differentiate between leucine and isoleucine, plus that the last position of a peptide is more restricted. This same argument carries over to decoy databases made by reversing peptides or proteins, as there is no biological or biochemical explanation for reversing an amino acid sequence; from this perspective, these decoy databases are "close to random". For short peptides, though, the overlap is not negligible: The target database of a human proteome contains some ten thousand tryptic peptides of length six; since the last position is restricted, several percent of reversed decoys of length six coincide with target peptides. For longer peptides, this overlap decreases rapidly, and it is practically zero beyond eight residues [101]. In practice, decoy peptides that coincide with a target peptide are removed from the decoy database (or shuffled again), as they would violate Assumption 1.

Claude helped!

What about Assumption 2? This cannot be verified empirically, because any decoy peptide we generate may, by chance, be present in the experimental sample we are looking at. Therefore, we have to rely on theoretical considerations: If the true peptide is in the decoy database then, by Assumption 1, it is not in the target database. This means that we have scored a "lucky punch": We were searching in a database of chicken proteins and just by chance, the true peptide, which is not from chicken, happens to be in the decoy database. But we have seen above that amino acid sequences in the decoy database are either random or "kind of random", so the probability of finding exactly the one we have in the sample is rather small and can be ignored.

As noted, Assumption 3 is the most important and, at the same time, the hardest to "discuss away". Empirically, we can generate a decoy dataset and show that bogus hits in the target database are as likely as hits in the decoy database: This is non-trivial, as we usually do not know which of the hits in the target database are bogus. But more importantly, this only "proves" that the assumption is satisfied for *this* target database and *this* list of query spectra; does it hold for any such combination? We can also approach the assumption from a theoretical standpoint; we will argue that some of the above methods to generate a decoy database are likely to fail the assumption, whereas others do not have such flaws. But even for those, empirical evaluation is required to see how good our FDR estimates are, see Sec. 6.7.

When we access the combined database, we filter candidates by the precursor mass of the query spectrum. To ensure that random (bogus) hits from the target database and hits from the decoy database have the same chance of coming out on top, a very natural request is that the same number of candidates comes from the target and the decoy database: By this, we provide "equal opportunities". In the extreme case, all query spectra have candidates exclusively from the target database; it is hard to argue that Assumption 3 still holds. Now, *inverted proteins*, *random iid* and *Markov chain* cannot guarantee that the number of candidates is identical; but all other methods can.

Next, the score of a peptide candidate depends crucially on the number of peaks the peptide can explain: Assuming ideal fragmentation (prefixes and suffixes only), the number of peaks is linear in the length of the peptide. None of the scores in Chapter 4 has been designed considering the subtleties of score comparability for peptides of different lengths. For this reason, it appears to be a reasonable idea to use decoys that mimic the length distribution of the target database, preferably for any query precursor mass. It turns out that all of the above methods to generate decoys except *inverted proteins* guarantee this in general, see Exercise 6.5; but *random iid* and *Markov chain* cannot guarantee this property for each individual query precursor mass.

Is there anything else we know about the peptides in the target database, which makes them substantially different from random peptides? In fact, there is one more thing: (Almost) all peptides in the

target database end with the characters K or R, and few of them have K or R at a different position. Why is this important for Assumption 3? Because (almost) all of our query spectra will correspond to peptides which also end with the characters K or R, because of tryptic digestion! This also holds for query peptides which are *not* in the target database. But if the query peptide and the target peptide end with the same character, then they already share two peaks in the ideal spectrum. Consequently, decoy peptides which have arbitrary characters at the last position, will have worse scores than random target peptides. Hence, we weed out all methods which do not preserve the peculiarities of the final position.

The methods that cannot be rejected based on theoretical considerations about Assumption 3, are therefore *pseudo-inverted peptides* and *pseudo-shuffled peptides*.

As a related topic, we have discussed in Sec. 5.1 why we cannot use spectra with random peak masses as reasonable decoys.

6.5 Searching separately in target and decoy database

The approach we have presented above searches in a database that combines targets and decoys. It is called *Target-Decoy Competition* (TDC), as each query results in a hit in either the target or the decoy database: Target and decoy peptides compete for being selected as the hit. A somewhat natural alternative is to search individually in the target database and in the decoy database, and to use search results to again estimate FDR. This is called *Target-Decoy Separate Search* (TDSS). Consider Fig. 6.1: Each query will now produce two hits, one from the target database and one from the decoy database.

The individual search has two conceptual advantages: The first is that we do not have to discard high-scoring hits, just because there has been a better hit in the decoy database. If we indeed believe that a high score of a hit is informative about it being correct or incorrect, then throwing away such high-scoring hits should make your heart ache. Statistics dictates that even the best and cleanest correct hit might get discarded: No matter how good the hit score is, there is a chance that a decoy scores better. The second advantage is that we can increase the size of our decoy database without having to go an extra mile in our statistical estimation method. We noted that q-value estimates are limited in their precision by the size of the batch we are searching; but maybe, the difference between 0 and 0.001 — the smallest non-zero q-value you can reach with 1000 queries — is very important to you. Separate search allows us to use a decoy database which is, say, 10 times the size of the target database. The necessary modifications for the estimation method are straightforward, but running times will increase substantially (say, 5.5-fold). To build a decoy database larger than the target database, we have to use one of the stochastic methods; in view of our discussion in Sec. 6.4, "*pseudo-shuffled*" is the method of choice.

Unfortunately, this approach also has a number of issues: It assumes not only that a bogus hit in the target database is as probable as a hit in the decoy database; it goes one step further and assumes that the distribution of scores is identical for these two sets. The scores discussed in Chapter 4 are not designed to guarantee this: If a query spectrum contains high-intensity noise peaks which cannot be explained by any peptide sequence, then scores of all peptides against this spectrum will be much lower than those of the same query spectrum without the noise peaks. Luckily, we already know a solution to that problem: Our score has to be calibrated, see Sec. 5.5. In contrast, target-decoy competition does not require us to use a calibrated score; I will not go into the details, but for large hit lists its estimates are (asymptotically) unbiased. For small lists, and if we pick the largest list whose estimated FDR is below our threshold, the estimate $k/(n-k)$ is slightly too optimistic; using $(k+1)/(n-k)$ instead provably controls the FDR [211]. But given that we assume throughout this whole chapter that high scores indicate high-quality hits, using calibrated scores appears to be a good idea anyway, even when using TDC.

Claude helped!

The second issue is more severe: Assume that we want to output the complete list of target hits; what is our FDR estimate for that? This implies that (basically) all our decoys also pass the score threshold; it does not have to be all of them, but it can be. Now, assuming that the number of bogus target hits above the score threshold equals the number of decoy hits above score threshold, we see that (almost) all of our target hits must be bogus! This is obviously nonsense. To correct for this issue, we have to introduce a prior

probability α_0 so that, for the complete list of hits, a hit is bogus with probability α_0. Our FDR estimate then becomes $\alpha_0 \cdot k/n$ where k and n are the number of decoy and target hits above the score threshold, respectively. But what is α_0? The conceptually simplest way to determine it uses target-decoy competition for the complete list, which does not require us to do any additional work: If, for k out of n queries, the decoy hit scores better than the target hit, then we expect another k bogus target hits among the remaining queries, so $\alpha_0 \approx 2k/n$; and whereas there are smarter, better ways, this shall do for us.

Unfortunately, it turns out that TDSS, with or without the correction via α_0, is not a good (unbiased) estimator for the false discovery rate. Luckily, there now exists a method called "mix-max" which solves this issue [176], combining unbiased estimates with the advantages mentioned above. Beyond the mandatory use of calibrated scores, its only disadvantage compared to TDC is that it is substantially more complicated and harder to explain; so, users will not like it.

6.6 Posterior error probabilities

As mentioned in Sec. 6.1 both FDR and q-value make a statement about the list of reliable identifications; what we want to know is the quality of each single hit.

The *Posterior Error Probability* (PEP) is simply the probability that an observed hit is incorrect. If the PEP associated with some hit (query plus candidate) is 2 %, then there is a 98 % chance that the candidate was in the mass spectrometer when the query was measured. The PEP can be thought of as a local version of the FDR; but whereas the FDR measures the error rate associated with a *collection of hits*, the PEP measures the probability of error for a *single hit*.

PEPs finally sound like what we are interested in: We can now decide — for each hit — whether it is correct or bogus! Why should we deal with significance estimates that tell us something only for a list of hits (FDRs, q-values), or deal with the unintuitive p-values? But unfortunately, PEPs and PEP estimation also have a number of "issues", as we will see below.

Since we are sorting hits by score, this means that PEP measures the error rate for hits with a given score T. To estimate a PEP, we have to estimate the number of correct hits a with score T, and the number of bogus hits b with score T: Then, the PEP is $b/(a+b)$. Compare this with FDR, where we counted the number of hits with score *at least* T. But since we have only a finite number of hits, there is usually *at most one hit* for any score T! Therefore, we have to model score distributions of true and bogus hits using some distribution. We can then estimate the likelihood a that score T was "generated" by a correct hit and the likelihood b that it was "generated" by a bogus hit; weighting both by the fraction of correct and bogus hits (the mixture weights, which are estimated alongside the distributions), we calculate the PEP again as $b/(a+b)$. Similar to p-value estimation in Sec. 5.3, we can model scores via certain parametric distributions; unlike there, we have to model not only the score distribution of bogus (random) hits, but also that of correct hits. To do so, we can use Expectation Maximization (EM) to *simultaneously* fit the two distributions for correct and bogus hits to the score distribution of the dataset. The mathematical details are more complicated than for FDR and q-value estimation, so I will stop at this point.

Advantages of PEPs over FDR and q-values include:

- Are you interested in one particular hit? For example, in a metabolomics sample, the first eight hits may be metabolites which you are not deeply surprised to observe in the dataset, whereas rank nine is a small sensation. In this case, the PEP of that metabolite is more relevant than its q-value.
- In almost all cases, you are not at all interested in the presence or absence of peptides in your sample; what you want to know is the presence or absence of (modifications of) proteins in your sample, see Chapter 13. To estimate how likely it is that a particular protein is present in your sample, PEPs of hits are conceptually the correct basis.

Now for the issues of PEPs and PEP estimation described above:

- We noted in Sec. 5.6 that modeling score distributions via parametric distributions has a number of issues; in particular, most score distributions are multimodal. This problem is further aggravated for PEP estimation because we have to model not only the score distribution for bogus identifications, but also that of correct identifications! Whereas it is rather easy to generate many false hits using decoys, it is much harder to generate many correct hits; so, our estimate of this distribution is less accurate.

- It is possible that for two hits, the hit with the better score gets the worse PEP estimate. This is somewhat counterintuitive but mathematically correct for the models we have established. This is not limited to the case that one of the distributions is multimodal, see Exercise 6.11. Unfortunately, this may be an artifact of our incomplete knowledge about the true distributions. We have observed a similar effect for FDR estimation; but there, we are returning sets of hits, and enlarging the set is welcome to us.

- One might naïvely assume that the PEP of a hit should not depend on the other hits in the dataset; but this is usually not the case: If we fit the score distributions via expectation maximization, then inserting high-scoring correct or bogus hits will obviously shift the corresponding distributions and, hence, change our estimates.

- Are you truly interested in one particular hit in a peptide database? Or, are you rather interested in a *set of hits* where we can guarantee that only a small fraction is wrong? Because this is exactly *not* what PEPs are about; if you are interested in identifying sets of hits, use q-values. If you set a threshold on the PEP of hits, then the list of hits passing the threshold will have an FDR below the chosen threshold; but the information you are missing is whether it is slightly below the threshold, or substantially below the threshold.

But PEPs finally allow us to estimate FDR while taking into account the *quality of each hit* in the list, see the end of Sec. 6.3: Assume that we have ordered all hits by posterior error probabilities $p_1, p_2, \ldots$, so $p_i \leq p_{i+1}$ — be reminded that this is not necessarily the same order as by hit score. Now, the expected number of bogus hits for a single hit i is p_i; simply think of flipping an appropriate coin. In technical jargon, this is a Bernoulli trial. Hence, the expected number of bogus hits among the first k hits is $\sum_{i=1}^{k} p_i$. Notably, we do not even have to assume that the coin flips (top k hits) are statistically independent; regarding expected values, $\mathbb{E}(X+Y) = \mathbb{E}(X) + \mathbb{E}(Y)$ holds whether or not random variables X, Y are independent. This allows us to estimate the false discovery rate of the top k hits as

$$\widehat{FDR}_k = \frac{1}{k} \sum_{i=1}^{k} p_i. \tag{6.3}$$

As long as we have a method to accurately estimate PEP, we can use this equation to estimate FDR taking into account the quality of *every* hit in the list. Note that FDR estimates via (6.3) are monotonically increasing, so $q_k = \widehat{FDR}_k$ is also the q-value estimate of hit k.

6.7 Evaluation of decoy databases and false discovery rate estimates

If you have generated a decoy database, you might wonder if it is of good quality. How can we test that? Our first evaluation is based on the fact that whenever you can estimate false discovery rates, you can also estimate p-values. For decoys, this was covered in Sec. 5.2: Our p-value estimate is $(k+1)/(n+1)$ if there are k decoys above the score threshold, and we considered a total of n decoys. But this is the p-value for an individual pair "query plus candidate"; for a hit, which is the best candidate from a set of candidates, we have to correct the p-value for multiple testing.

If we have a set of reference queries where we know the correct answer, we can score each query against any incorrect candidate, compute the p-value of the score using the decoys, and collect these

computed p-values. We do not have to correct p-values, as we are considering p-values of individual queries, not hits. Alternatively, we can estimate p-values for hits (best-scoring candidates); in this case, we have to correct p-values as $1-(1-p)^N$ if there are N candidates and p is the p-value of the highest-scoring pair "query plus candidate". Note that we cannot use the simple correction $N \cdot p$ as the corrected p-value may become large, and $N \cdot p$ may even be larger than one.

Under the null model, p-values follow a uniform distribution: We can visualize that by a histogram plot of the p-values. In a uniform plot, all bars in the histogram have roughly the same height. Alternatively, we can sort the p-values, then plot the (relative) rank of a p-value against its value. For a uniform distribution, this should be close to a straight line.

Unfortunately, we have to use reference spectra as queries to execute the above evaluation. One can argue that real query spectra can look very different from these reference spectra; also, we may not be satisfied with the number of available reference spectra as queries. For this reason, a more involved route is to use *entrapment* candidates: These candidates are much like decoys; the difference is that we do not use them for significance estimation, but for evaluation of such estimates. For example, we can use peptides from an evolutionarily distant organism (see Sec. 6.8) to build the entrapment database. We search in a combined database consisting of our true peptide targets (proteins from the organism our sample comes from, plus the usual contaminant proteins such as keratin) plus the entrapment targets (from a distant organism). For all hits in the entrapment database, we know that the answer must be wrong! Note that entrapment hits are only a lower bound for the number of bogus hits; we have to scale them by the size ratio of entrapment and target database to also account for bogus hits from the target database.

We can sidestep estimating the uncorrected p-value "query plus candidate" and instead, use *only the hits* in target and decoy database under target-decoy competition for p-value estimation. We estimate the p-value of a target hit with some score T as $p' = (k+1)/(n+1)$ where k is the number of decoy hits with score T or larger, and n is the total number of decoy hits. To reach meaningful p-values requires a very large number of (decoy) hits; hence, the approach is usually not suited for reference datasets.

The above evaluation is not limited to decoy databases but instead, can and should be performed for any method that claims to do FDR or p-value estimation.

But we also want to evaluate our FDR estimates directly — how can we do this? We have seen that false discovery rates go up and down as we increase the score threshold; so we rather compare q-values. We again assume a set of reference queries where we know the correct answer. We search this set in some database and generate a list of hits, which we sort by score. As we know the correct answers, we can compute the exact q-values as described in Sec. 6.1. But we can also ignore that we are omniscient, and estimate q-values as described in Sec. 6.3. Now, every hit $i = 1, \ldots, n$ has an exact q-value q_i and an estimated q-value $\widehat{q}_i$ attached to it. We plot these pairs $(q_i, \widehat{q}_i)$ in a scatterplot. This is sometimes referred to as a quantile-quantile (Q-Q) plot, which is not incorrect, as both the q_i and the $\widehat{q}_i$ are non-decreasing, $q_i \leq q_{i+1}$ and $\widehat{q}_i \leq \widehat{q}_{i+1}$.[1] But the important point is: If our estimates are of high quality, then the points should be close to the line $x = y$. As we are particularly interested in small q-values, we can also plot pairs $(\log q_i, \log \widehat{q}_i)$ — or, even better, plot $(q_i, \widehat{q}_i)$ on a logarithmic scale.

Have a look at Fig. 6.2, taken from Scheubert *et al.* [311] where an FDR estimation method for small molecule library search is presented. I have deliberately chosen a method which I have co-developed myself, not because of vanity, but because I want to play "blame and shame". The details of the method are of no importance to us; the only relevant question is: Are false discovery rate estimates in Fig. 6.2 of "good quality"? The p-value distribution plot looks reasonable; it is far from uniform, but maybe that is OK. But if we have a look at the Q-Q plot, we can get a clear understanding of what we can trust and what we cannot with regard to the estimates. In detail, the Q-Q plot compares three different methods for generating decoys; two methods reach similar estimates whereas the naïve method is so inaccurate that it is of no use in practice, see Sec. 6.2. For the two "real" methods, estimates are far off the true

[1] If I am not mistaken, we can also interpret this as a probability-probability (P-P) plot if we think of the q-values (exact and estimated) as the cumulative distribution function of the score. Is all of that correct? Statisticians! This is probably too much information, though.

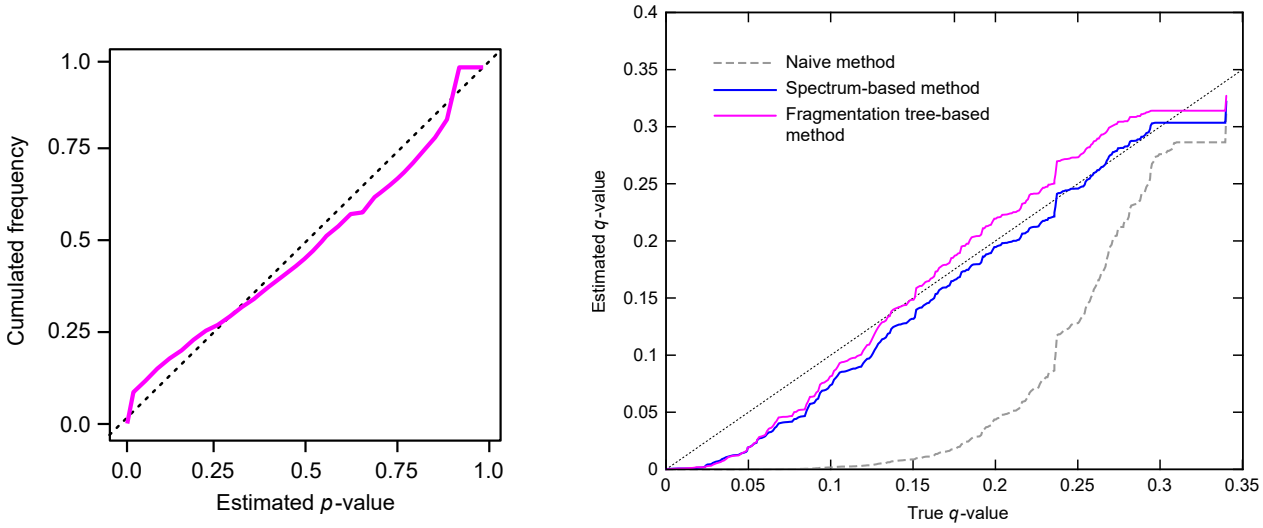


Figure 6.2: Exemplary evaluation of false discovery rate estimation using a decoy database. Left: Estimated p-values vs. cumulative frequency ("true p-value"). Right: Estimated vs. true q-value; three methods (including a naïve method) are compared. The naïve method is obviously not suggested for q-value estimation; rather, we want to know how bad estimates are in practice. Figure taken from [311].

values in particular for small q-values: When the true q-value is 2 %, the estimated q-value is still close to zero. In comparison to what people are used to in shotgun proteomics and peptide identification, FDR estimates are ridiculously bad… It turns out that estimation inaccuracies are not due to our inability to build good-quality decoy databases, but rather an intrinsic problem of FDR estimation in metabolomics, see Sec. 6.8. The q-value estimation in Fig. 6.2 allows us to evaluate the severity of the problem, and to give reasonable suggestions to the user, such as: q-value estimates below one percent must not be trusted; rather expect 3 % false identifications for those. The danger here is that *biologists and chemists will forget about this fact*; when users see a substantial number of hits with estimated FDR 1 % they will report them — and, potentially believe them — as such. In particular, users will neither understand nor approve if we improve our method and estimates get "worse", that is, closer to the real values. But inaccurate or not, this is substantially more information than not estimating an FDR at all, or simply *claiming* that some measure is a p-value or q-value without this evaluation!

6.8 The limits of decoy databases: Metaproteomics and metabolomics

Everything is good in shotgun proteomics as long as you are considering the proteome of a single organism, or the proteomes of a few organisms which, in the best of worlds, are not closely related. But when we go to *metaproteomics*, where we want to study the proteins from a sample of an environmental source, things get ugly. The funny thing is: Everything is fine with our decoy database; it is the target database that is causing problems.

Let us assume we are studying a sample from some microbial community; our target database consists of peptides from 100 microbial proteomes. We build our decoy database as described above, using the pseudo-inverse or pseudo-shuffle method. If you do so, you will find that many bogus hits will receive excellent q-value estimates, and that your FDR estimates are much too optimistic. In truth, I have never done this experiment; I simply claim it to be true, based on the observation that our *target database* is no longer suitable for FDR estimation.

What do I mean by that? When we built our decoy database, we were a little imprecise when we discussed Assumption 3: The assumption says that "a bogus hit in the target database is as likely as a hit

in the decoy database". But what we discussed afterward was whether a *random hit* in the target database is as likely as a hit in the decoy database! We did this without blushing because the target database is extremely *sparse*: The target database contains several hundred thousand peptide sequences, which is basically nothing in comparison to the $1.08 \cdot 10^{13}$ peptides of length up to ten. Furthermore, two peptide sequences of reasonable length in the proteome of a single organism with edit distance at most, say, 3 exist — we will see an example below — but they are a small minority of all peptides. Hence, any bogus hit in the target database is indeed as good or bad as a hit against a random peptide sequence.

But if we consider a target database from multiple organisms, this is no longer true: The different taxa have orthologous proteins, and for reasonably closely related organisms, the protein sequence will be similar. This implies that we also have peptides in our target database with similar sequence. Let us consider one particular case of what these similar peptides may look like: We assume that the two peptide strings are identical, except for two consecutive characters (amino acids) which have been swapped. For example, one string may be PEPTIDE whereas the other one is PETPIDE. (If you insist on tryptic peptides, feel free to add an R to the end.) If your query spectrum is that of PEPTIDE, then the PETPIDE candidate will receive a much higher score than you would expect for a random string.

In fact, the problem is much worse than that. Above, we implicitly assumed that we have both strings PEPTIDE and PETPIDE in our target database. But in practice, it is more likely that only one of the two (say, PETPIDE) is present in our target database — because the genome of the organism with the corresponding protein has been sequenced — whereas the other is absent, simply because the corresponding organism has not been sequenced. This is more likely as only a small fraction of organisms in our sample have been sequenced, and this is not going to change quickly. Now, assume we have the PEPTIDE string from the unsequenced organism in our sample. We will then find an excellent agreement of its query spectrum with the PETPIDE string in the target database, reaching a hit score which is hard to explain by some random process. Et voilà: Here is a hit which will presumably receive a very good q-value but which is wrong.

I have chosen the case of two consecutive, swapped amino acids because it is easy to understand that the resulting tandem mass spectra are very similar: Only the peaks for prefixes PEP vs. PET and suffixes TIDE vs. PIDE change. As soon as you have understood that, you can build yourself numerous other types of string transformations that will cause basically the same problem, in any shade and degree of the problem you can think of. This explains why we cannot simply ignore the problem by saying, "a peptide with only two swapped amino acids is not a severe problem for me and my application".

The fact that this is a problem of the target database and not of the decoy database, means that we cannot solve the problem by building a different, "smarter" decoy database. Hopefully, one day a smart statistician will solve this problem, or "discuss it away". Until then, we have to deal with the fact that small q-value estimates can be accurate only if our target database is not too "clogged".

Similar things can happen *within* one organism: For example, consider the proteins Actin (ACTB) and Beta-actin-like protein 2 (ACTBL2) in human, which contain the tryptic peptides VAPEEHPVLLTEAPLNPK and VAPDEHPILLTEAPLNPK, respectively. These two peptides differ only at positions 4 (E vs. D) and 8 (V vs. I). Now, the molecular formula of a glutamic acid residue plus a valine residue is $C_5H_7NO_3 + C_5H_9NO = C_{10}H_{16}N_2O_4$, and so is that of an aspartic acid residue plus an isoleucine residue. Hence, these peptides have exactly the same mass — as long as we ignore binding energies, as we have done throughout this book — and the fragmentation spectra of these peptides share a large number of peaks, much larger than what you expect by chance from a random match. But this is a *rare* thing and, hence, will not interfere substantially with FDR estimation; the variation in FDR estimation through the actual data you look at is potentially much larger.

Exactly the same problem makes FDR estimation hard when you are doing metabolomics, see Sec. 10.1. (It is somewhat funny that both areas with "FDR estimation issues" start with "meta".) Here, the set of all known metabolites is even smaller in comparison to the space of all possible molecular structures, see Sec. 10.4. Unfortunately, metabolite structures also tend to "flocculate": Two structures where we move a hydroxyl group (OH) from one carbon atom to an adjacent carbon atom, are often both valid metabolites. As above, these two structures can have basically indistinguishable fragmentation patterns. Again, our

knowledge of all metabolites is vastly incomplete, so we will usually not even notice if we have identified the wrong one.

In fact, the situation is even worse for metabolites: It is a highly non-trivial problem how to simulate the tandem mass spectrum of a metabolite if we only know its structure, see Chapter 10; and, it is unclear how to generate decoy structures, as small modifications of existing metabolites may be true metabolites whose structure has not been elucidated so far, whereas large modifications may result in decoys which are too different from our target structures, see Chapter 5.

6.9 Historical notes and further reading

My presentation of decoy databases is inspired by Elias and Gygi [101] and Keich, Kertesz-Farkas and Noble [176]; the latter paper gives an excellent introduction to the different paradigms in FDR estimation. See also the review by Nesvizhskii [250]. I want to stress that the below coverage of literature on FDR estimation is in no way comprehensive.

In this chapter, we have talked about FDR, q-values and PEPs of *hits*, not of peptides. We have deliberately done so, as multiple hits may point to the same peptide; some of these hits may be correct, others may be bogus. You have to correct for that if you want to make statements about the peptides present in your sample. Keep in mind that in the end, we are interested in proteins and not peptides (Chapter 13).

Historically, the first method for estimating error rates for a batch of hits in shotgun proteomics is due to Keller, Nesvizhskii, Kolker and Aebersold [181] in 2002: PeptideProphet did not estimate false discovery rates but rather posterior error probabilities. Furthermore, it did not use decoys but rather empirical Bayes estimation, where expectation maximization is used to fit the mixture distribution of (presumably) correct and (presumably) bogus hit scores. Later versions of PeptideProphet [57, 58] integrated decoys into the estimation procedure. It is also possible to estimate PEPs using non-parametric regression [5, 98, 170, 339]. See Käll *et al.* [169] on the relationship of q-values and posterior error probabilities.

Decoy databases were used for peptide identification as early as 2002 [243, 269]; target-decoy competition was described in detail and evaluated by Elias and Gygi [101], whereas target-decoy separate search was analyzed by Käll, Storey, MacCoss and Noble [168]. Elias and Gygi [101] also empirically supported Assumptions 1 and 3 for TDC, and argued that violations of Assumption 2 are exceedingly rare.[2] See Keich and Noble [178] on the importance of calibrated scores even if you are using TDC. Using larger decoy databases for TDC is, as mentioned, a non-trivial problem; Keich *et al.* [180] show how it can be done. The mix-max method is due to Keich *et al.* [176]; see also there for empirical results that the common variants of TDC and the original TDSS (with or without prior α_0) are not unbiased estimators, whereas mix-max is; the only unbiased TDC variant forfeits a random subset of high-scoring hits.

Transforming PEPs into FDR using (6.3) is the Averaging Theorem (4.4) by Efron and Tibshirani [97], but they call PEPs "local FDR".

Keich and Noble [179] suggest an alternative route of FDR estimation, which is based on correcting p-value estimates similar to Benjamini-Hochberg [18]. It explicitly takes into account that there are two types of false identifications, one being due to foreign spectra — which cannot be found in our incomplete database — the other being due to wrongly assigned spectra. On the downside, estimates can only be as good as the initial p-value estimates.

Calibrated scores were suggested by Jeong, Kim and Bandeira [164] as "normalized scores"; this was formalized by Keich and Noble [178] who suggest using shuffled peptides for this purpose. With regard to the importance of evaluating your FDR and q-value estimates, Jeong *et al.* [164] showed that, depending on the TDA variant and search setup, true false discovery rates can be more than 10-fold higher than the estimated ones.

[2] The protocol in [101] suggests also reporting the decoy hits to the user and using $2k/n$ as the FDR estimate; in view of Assumptions 1 and 2, this is extremely funny advice: "Here are your search hits; some hits are not from your database and definitely wrong, but whatever!" — I like the spirit.

The Actin example was suggested by Brian C. Searle. For metabolomics, generating decoy metabolite structures as well as simulating tandem mass spectra remain open problems. Scheubert *et al.* [311] sidestep the problem by transforming target *spectra* into decoy spectra, without ever considering the structure of the target metabolite. But this trick cannot sidestep the fundamental problem that bogus hits in the decoy database are often not random (Sec. 6.8), resulting in overly optimistic FDR estimates.

The story about Charlie Chaplin is one of these stories that are too nice to be true, because it is not true — despite numerous people having reported it in books and articles [241]. Unfortunately, it is most probably an urban myth that spread virally in some of the 1920s gossip columns.[3] It nevertheless contains a grain of truth: Sometimes, something is more similar to the real thing than the real thing itself.

6.10 Exercises

6.1 When calculating FDR exactly, why do we have to consider only sublists consisting of the top k hits from the sorted list, for any $k = 0, \ldots, n$?

6.2 For a list of k bogus hits and n total hits, what is the smallest non-zero q-value any hit can reach?

★ 6.3 What is the smallest non-zero *difference* between any two q-values in a list with n hits?

6.4 Show that the smallest non-zero difference between any two q-values in a list with n hits, is at least $1/n^2$.

6.5 Assume that we build a decoy database using the "inverted proteins" method. Explain why we cannot guarantee that the decoy database contains exactly the same number of peptides; or, that the distribution of peptide lengths in the two databases is identical. One protein does the trick.

6.6 Given a target database of proteins

$$\{\mathsf{TVKQDEGHRWTL}, \mathsf{YPPNKCRRDHIKVRAA}, \mathsf{DDCDKPKMN}, \mathsf{FIKTTSRQPRVYYC}, \mathsf{MNMQKWAWAKFIFIRVW}\},$$

build the corresponding peptide decoy databases for methods "inverted proteins", "inverted peptides", and "pseudo-inverted peptides".

6.7 For the target database from the previous exercise, build the "random iid" model with pseudocounts (any character receives 0.5 observations as a starter package).

★ 6.8 For the target database from Exercise 6.6, build the "Markov chain" model of order 2 with pseudocounts (any two-letter string receives 0.5 observations).

6.9 We want to do target-decoy separate search, and have computed the hit scores of query i against the target database (t_i) and the decoy database (d_i), for $i = 1, \ldots, n$. Explain how we can use target-decoy competition to estimate the prior probability α_0 that a hit in our list of hits is bogus. Use only known values t_i and d_i.

★ 6.10 Enlarging the decoy database results in more accurate and more robust q-value estimates, but comes at the price of substantially increased running times. Develop an adaptive strategy to control this overhead.

6.11 Assume that the score distribution of correct hits is $\mathcal{N}(3,1)$, the score distribution of bogus hits is $\mathcal{N}(0,2)$, and 10 % of the hits are correct. In what intervals of the real axis is the posterior error probability below and above 50 % and 1 %, respectively?

6.12 Show that for numbers $a_1, a_2, \ldots$ with $a_i \le a_{i+1}$, the averages $b_k = \frac{1}{k}\sum_{i=1}^{k} a_i$ also satisfy $b_k \le b_{k+1}$.

[3] https://skeptics.stackexchange.com/questions/9423/

7 Isotope Distributions and Isotope Patterns

> *"Two very significant discoveries are due to mass spectroscopic studies. First, J.J. Thomson discovered that neon consisted of a mixture of two different isotopes (masses 20 and 22) rather than only a single isotope. This observation of the existence of stable isotopes is perhaps the greatest achievement that can be claimed by mass spectroscopy. [...] The second significant discovery due to mass spectrographic studies was made by F.W. Aston. He observed that the masses of all isotopes are not simple multiples of a fundamental unit, but rather they are characterized by a mass defect; i.e., isotopes do not have integral masses."* (Robert W. Kiser, *Introduction to Mass Spectrometry and Its Applications*)

Mass spectrometry cannot detect single molecules, but is dependent on the existence of millions of "identical" copies of some molecule. These copies are identical from a chemical standpoint, but not from a physical standpoint: Throughout these copies, elements follow their natural isotope abundances. For mass spectrometry, this implies that instead of a single peak, we observe an isotope pattern of the molecule. On the one hand, this is simply an additional complication that we have to deal with when analyzing MS data. In many MS applications, the experimental setup is actually chosen so that we do not have to consider such isotope patterns: In small-molecule analysis (Chapter 9), one deliberately selects only the monoisotopic peak (see below) for fragmentation, and no isotope patterns can be observed in the fragmentation spectrum. In other applications such as peptide *de novo* sequencing (Chapter 2), we usually try to get rid of isotope peaks during preprocessing, a process called "deisotoping". On the other hand, we can use this fact to our advantage: Namely, we can use the isotope pattern to derive information about an unknown molecule, namely its molecular formula. This will be addressed in Chapter 8.

Although in principle, each and every molecular formula should correspond to *some* molecule, our formalism does not distinguish between reasonable molecular formulas (such as $C_{12}H_{22}O_{11}$) and unreasonable molecular formulas (such as CH_{37}). For the sake of readability, we will use unreasonable molecular formulas (such as H_{100}) in our examples and theoretical considerations whenever this leads to simpler calculations. Such examples might provide the reader with a rough estimate of, say, the required size of a molecule. For this purpose, an unreasonable molecular formula should do the job. We will come back to this point in Sec. 8.4, where we reject molecular formulas that cannot correspond to some molecule. Trying to integrate such chemical knowledge at a low level, will usually destroy both the comprehensibility and the swiftness of our methods. Instead, chemical knowledge should be integrated at a higher level, such as rejecting molecular formulas after they have been enumerated.

7.1 Isotopes

We continue our journey into the realm of physics that we have started in Sec. 1.1. We briefly recall some of the facts from there: Atoms are composed of electrons with a negative charge, protons with a positive charge, and neutrons without charge. Protons and neutrons make up the atomic nucleus. Atoms have no charge, whereas charged particles are called ions. Atoms are classified by the number of protons in the atom, which defines which element the atom is. Atoms with identical atomic number cannot be differentiated chemically. Elements most abundant in biomolecules are hydrogen (H, atomic number 1), carbon (C, 6), nitrogen (N, 7), oxygen (O, 8), phosphorus (P, 15), and sulfur (S, 16). The "backbone" of all biomolecules is made from carbon, and we often classify elements based on their similarity or dissimilarity to carbon. Less abundant elements include boron, fluorine, silicon, chlorine, copper, zinc, and selenium, see Table 7.5.

The *nominal mass* or *nucleon number* of an atom is its total number of protons and neutrons. An element can have numerous different atoms with an equal number of protons and electrons, but a varying number of neutrons. These are called *isotopes* of the element. The nucleon number is denoted in the upper left corner of an atom, such as ^{12}C for the carbon 12 isotope with 6 protons and 6 neutrons. Several isotopes of an element can be found in nature and are called *natural* isotopes. The natural isotope with the lowest mass is called *monoisotopic*, such as ^{1}H, ^{12}C, ^{14}N, ^{16}O, ^{31}P, and ^{32}S. As an example, the relative abundance of the monoisotopic carbon isotope ^{12}C is 98.93 %, whereas the isotope ^{13}C has a relative abundance of about 1.07 %. The radioactive isotope ^{14}C with half-life 5730 years has a relative abundance of less than 0.001 % in nature, and is usually ignored in our analysis; likewise, we can ignore tritium ^{3}H.

A short discussion is in order with respect to the term "monoisotopic"; if you are from bioinformatics or computer science, you can safely skip this paragraph. The International Union of Pure and Applied Chemistry (IUPAC) made the unfortunate decision to define the monoisotopic mass of a molecule as "the sum of masses of the atoms in a molecule (or ion) using the unbound, ground-state, rest mass of the most abundant isotope for each element." (IUPAC forgot to define what the monoisotopic mass of an atom is, which is somewhat odd, as the molecular formula H_1 *has* a monoisotopic mass using their definition; furthermore, it makes the definition unnecessarily complicated, sacrificing heredity of the property for no reason.) Note that this definition results in differences to the one above only for a few elements relevant for biomolecules (none from Table 7.1) such as selenium or boron, see Table 7.5. But as soon as you start working with isotope patterns computationally, you will notice that the IUPAC definition is not helpful at all: Using the IUPAC definition, the monoisotopic mass is often not the most abundant isotopologue (see below) of the molecule, it is often not resolved from other isotopologue peaks, and it may be undetectable in an MS experiment as it has intensity below noise level. In particular, given the experimental isotope pattern of an unknown molecule, it is generally impossible to determine which of the peaks corresponds to the monoisotopic peak![1] Using our definition, the monoisotopic mass of a molecule is always the sum of monoisotopic masses of the atoms; the monoisotopic peak is in all cases the first peak of the ideal isotope pattern; and, the monoisotopic (isotopologue) peak is always resolved from all other isotopologue peaks, even at unit mass accuracy. Clearly, the monoisotopic peak of a molecule may again be undetectable in an MS experiment; this is mainly an issue for molecules beyond 3000 Da and when "uncommon" elements have to be considered, see Exercise 7.7 and Sec. 8.7. In the computational mass spectrometry literature, the difference between the two definitions is usually ignored, as it is of relevance only if your molecules contain "uncommon" elements such as selenium or boron; it is implicitly assumed that, say, the monoisotopic mass of the molecule is the first peak detected in its isotope pattern. According to IUPAC, that is wrong.

It is important to notice that unlike other numbers in this section, abundances of natural isotopes are not physical constants: These abundances vary depending on time and place (continent, planet, solar system) where the sample is taken. In fact, physicists may determine the origin of a sample based on its isotope abundances. For example, deuterium (^{2}H) varies in relative abundance from about 0.012 % to 0.016 % in non-marine organisms [78]. For computational mass spectrometry this is usually irrelevant; we just keep in mind that isotope abundances are not an "exact science" like masses. Regarding the six elements most abundant in living beings, see Table 7.1 for a detailed list of all natural isotopes and their relative abundance in nature. Isotopes not listed here have relatively short half-lives and, hence, are not found in nature at significant levels. At the end of this chapter, Table 7.5 on page 126 provides the same information for "less frequent" elements. Note that ionic bonds are not stable enough to be seen in MS, and metal ions may "fall off" the molecule; hence, these are not fully covered in Table 7.5. Exceptions are organometallics containing covalent metal-carbon bonds, such as organomercury and organotin compounds which are used as fungicides and antifouling agents and, hence, are important in environmental analysis [236]. Other exceptions are coordination complexes such as heme, where a central

[1] There is another noteworthy problem with the IUPAC definition: It is only valid for planet Earth, and might not even be unambiguous there. But on other planets, moons or asteroids, abundances of isotopes are different. For example, ^{81}Br can easily become monoisotopic instead of ^{79}Br.

element (symbol)	isotope	abundance %	mass (Da)	atomic weight (Da)
hydrogen (H)	^{1}H	99.988 %	1.007825	
	^{2}H	0.012 %	2.014102	1.008
carbon (C)	^{12}C	98.93 %	12.0	
	^{13}C	1.07 %	13.003355	12.011
nitrogen (N)	^{14}N	99.636 %	14.003074	
	^{15}N	0.364 %	15.000109	14.007
oxygen (O)	^{16}O	99.757 %	15.994915	
	^{17}O	0.038 %	16.999132	
	^{18}O	0.205 %	17.999160	15.999
phosphorus (P)	^{31}P	100 %	30.973762	30.973762
sulfur (S)	^{32}S	94.99 %	31.972071	
	^{33}S	0.75 %	32.971459	
	^{34}S	4.25 %	33.967867	
	^{36}S	0.01 %	35.967081	32.06

Table 7.1: Natural isotope abundances: Relative abundance of isotopes and their masses in Dalton, for the six elements most abundant in biomolecules. Masses are rounded to six decimal places. In addition, atomic weights of the elements are given.

atom or ion (an iron ion for heme) is surrounded by an array of ligands. Heme is sometimes covalently bound to a protein, in which case you may see the Fe ion even in protein mass spectrometry.

Recall that 1 Dalton is 1/12 of the mass of one atom of the ^{12}C isotope, so

$$1\,\text{Dalton} \approx 1.660538 \cdot 10^{-24}\,\text{g} \quad \text{and} \quad 1\,\text{g} = N_\text{A}\,\text{Dalton}$$

where $N_\text{A} \approx 6.022141 \cdot 10^{23}$ denotes the Avogadro constant. Also recall that due to nuclear binding energy, an atom's mass is smaller than the sum of masses of the contained protons, neutrons, and electrons; this is the mass defect. For example, the mass of a proton is 1.00728 Da, the mass of a neutron 1.00866 Da, and the mass of an electron is about 0.00054 Da. So, 6 protons, 6 neutrons, and 6 electrons have a total mass of 12.09888 Da whereas the corresponding ^{12}C atom has a mass of exactly 12 Da, a deviation of about 0.8 %. See Table 7.1 above for the masses of isotopes of the elements most abundant in living beings. Throughout this book, masses are usually rounded to six decimal places. It must be understood that this is done for readability; do not use these numbers if you are writing a computer program!

The *atomic weight* of an element is the expected mass (weighted average) over the natural distribution of isotopes. For example, the average mass of nitrogen is 0.99636 times the mass of ^{14}N plus 0.00364 times the mass of ^{15}N. See Table 7.1 for atomic weights of elements most abundant in living beings. Due to the variation of isotope abundances, average masses are not physical constants and depend on time and place where the measurement is taken. This is why atomic weights in Table 7.1 are given with fewer digits.[2]

A *molecule* consists of a stable system of two or more atoms. The *molecular formula* tells us the number of atoms that compose the molecule, and can be thought of as a compomer over the set of elements. Molecules with the same atoms in different arrangements are called *isomers*. For example, the chemical formula $(CH_3)_3CH$ implies a chain of three carbon atoms, with the middle carbon atom bonded to another carbon, and the remaining bonds connected to hydrogen atoms. We will not use chemical formulas in this textbook, so we do not have to define them formally. In comparison, the molecular formula C_4H_{10} only tells us that the molecule is made up of 10 hydrogen and 4 carbon atoms: A straight line of (single bond) carbon atoms with remaining bonds leading to hydrogen atoms has an identical molecular

[2] https://www.ciaaw.org/atomic-weights.htm

^{12}C	^{13}C	^{1}H	^{2}H	^{16}O	^{17}O	^{18}O	nom. mass	mass (Da)	abundance (%)
12	0	22	0	11	0	0	342	342.116215	85.3428
11	1	22	0	11	0	0	343	343.119570	11.0765
12	0	22	0	10	1	0	343	343.120432	0.3576
12	0	21	1	11	0	0	343	343.122492	0.2253
12	0	22	0	10	0	1	344	344.120460	1.9292
10	2	22	0	11	0	0	344	344.122925	0.6589
11	1	22	0	10	1	0	344	344.123787	0.0464
12	0	22	0	9	2	0	344	344.124649	0.0007
11	1	21	1	11	0	0	344	344.125847	0.0292
12	0	21	1	10	1	0	344	344.126709	0.0009
12	0	20	2	11	0	0	344	344.128769	0.0003

Table 7.2: Isotopologues of sucrose molecules $C_{12}H_{22}O_{11}$, sorted by mass. Isotopologues with nominal mass 345 and above omitted.

formula and, hence, is an isomer of the previous molecule. Molecules can have a net electric charge, that is, more electrons than protons or vice versa, and such molecules are called *ions*.

The nominal mass of a molecule is the sum of protons and neutrons of the constituting atoms. The mass of a molecule is the sum of masses of the atoms it is composed of. Here, a warning seems to be in order: The energy of a molecule is smaller than the energy of the constituting atoms due to the chemical bonds and intramolecular bonds in the molecule. According to $E = mc^2$, the mass of a molecule shows yet another mass defect through its structure, and is slightly smaller than the mass calculated above. But this mass defect is several orders of magnitude smaller than the atom mass defect, and can be safely ignored in all of our calculations. So, in a very strict sense, isomers do not necessarily have identical mass; but they do, as far as we are concerned.

Clearly, the mass and nominal mass of a molecule depend on the isotopes that constitute it. Accordingly, the *monoisotopic mass* of a molecule is the sum of masses of the constituting atoms where for every element, we choose the monoisotopic isotope. For example, sucrose $C_{12}H_{22}O_{11}$ has monoisotopic mass $12 \cdot 12.0 + 22 \cdot 1.007825 + 11 \cdot 15.994915 = 342.116215$ Da and monoisotopic nominal mass $12 \cdot 12 + 22 \cdot 1 + 11 \cdot 16 = 342$ Da. The *molecular weight* of a molecule is the sum of atomic weights of the constituting atoms. (Sigh.) If x is the molecular weight of your molecule in Dalton, then x g of pure substance contain N_A copies of the molecule.

7.2 Isotope distributions and isotope patterns

Mass spectrometry generally cannot detect single molecules but, just like most analysis techniques in life sciences, is dependent on the existence of millions of identical copies of some molecule. This means that elements follow the natural isotope abundances from the previous section: Instead of identical copies, we have different *isotopologues* of a molecule. For example, $^{12}C_{12}\,^{1}H_{22}\,^{16}O_{11}$ and $^{12}C_{9}\,^{13}C_{3}\,^{2}H_{22}\,^{16}O_{7}\,^{17}O_{3}\,^{18}O$ are two isotopologues of sucrose $C_{12}H_{22}O_{11}$. The *mass* of an isotopologue is the sum of masses of the constituting isotopes. The isotopologue where each atom is the isotope with the lowest nominal mass is called *monoisotopic*. See Table 7.2 for the first eleven isotopologues of sucrose.

The number of distinct isotopologues of a molecule is

$$\text{number of isotopologues} = (i_C + 1)\,(i_H + 1)\,(i_N + 1)\binom{i_O + 2}{2}\binom{i_S + 3}{3} \tag{7.1}$$

where i_E denotes the multiplicity of element E in the molecule, $E \in \{C, H, N, O, P, S\}$. This follows because for an element E with r natural isotopes, a molecule E_l consisting of l atoms of the element has $\binom{l+r-1}{r-1}$

nominal mass	342	343 (+1)	344 (+2)	345 (+3)	346 (+4)	347 . . . 398
abundance %	85.3428	11.6595	2.6656	0.2913	0.0371	< 0.004

Table 7.3: Isotope distribution of sucrose $C_{12}H_{22}O_{11}$ in percent, rounded to four decimal places.

different isotopologues. Note that $\binom{n}{0} = 1$ for all $n \in \mathbb{N}$. For example, sucrose has $13 \cdot 23 \cdot \binom{13}{2} = 23\,322$ isotopologues.

Mass spectrometry is often not capable of resolving isotopologues with identical nominal mass; instead, these isotopologues may appear as one peak in the MS output. Clearly, this depends on the resolution of the instrument *and* the analyzed molecule: Analyzing a molecule that contains sulfur with high-resolution MS, we may see two peaks for monoisotopic nominal mass plus 2. The same is true for other elements whose isotope mass differences differ significantly from that of carbon. See Sec. 7.5 for more details. Finally, ultra-high resolution instruments may resolve more isotopologues. For the moment, we simply ignore this fact.

If the nominal mass of an isotopologue is *significantly* larger than the monoisotopic nominal mass, then isotopologues with distinct nominal masses may have almost identical real masses. Consider the molecular formula $C_{345}H_{344}$ with nominal monoisotopic mass 4484: The isotopologue $^{13}C_{345}{}^{1}H_{344}$ has nominal mass 4829 and mass 4832.849275 Da whereas the isotopologue $^{12}C_{345}{}^{2}H_{344}$ has nominal mass 4828 and mass 4832.851088 Da. But the relative abundance of these isotopologues is so small that they are not detectable by mass spectrometry: In fact, there will usually be zero copies of these isotopologues present in our sample, see Exercise 7.1. Hence, we may safely ignore this subtlety.

We merge isotopologues with identical nominal mass; we refer to the resulting distribution as the molecule's *isotope distribution* (or *isotopic distribution*). How can we formally model this isotope distribution? For each element $E \in \Sigma$ we define a discrete random variable, denoted Y_E, representing the nominal mass distribution of the element. For example, Y_C with state space $\{12, 13\}$ and

$$\mathbb{P}(Y_C = 12) = 0.98930, \quad \mathbb{P}(Y_C = 13) = 0.01070$$

is the random variable of carbon, whereas Y_O with state space $\{16, 17, 18\}$ and

$$\mathbb{P}(Y_O = 16) = 0.99757, \quad \mathbb{P}(Y_O = 17) = 0.00038, \quad \mathbb{P}(Y_O = 18) = 0.00205$$

is the random variable of oxygen.

Now, the random variable Y of a molecule is the sum of random variables of the atoms constituting the molecule, where we choose these random variables according to the element of each atom. Unfortunately, we have to deal with a subtlety in the stochastic notation: We cannot write $Y_{H_2O} = Y_H + Y_H + Y_O$ for the isotope distribution of H_2O, as this would not result in two independent random variables for hydrogen but instead, one random variable whose value is doubled. Hence, we have to go a slightly longer road. We write $Y \sim Y'$ if two random variables are *independent identically distributed*. So, $\mathbb{P}(Y = y) = \mathbb{P}(Y' = y)$ holds for all y in the state space, but Y and Y' are independent. Given a molecule consisting of l atoms, we assign to each atom i a random variable Y_i, for $i = 1, \ldots, l$, such that $Y_i \sim Y_{E_i}$ where E_i is the element of the ith atom. Now we can represent the molecule's *isotope distribution* by the random variable $Y := Y_1 + \ldots + Y_l$.

Example 7.1. Consider sucrose with molecular formula $C_{12}H_{22}O_{11}$. The isotope distribution of sucrose is the random variable $Y = Y_1 + \cdots + Y_{45}$ where

$$\begin{aligned} Y_i &\sim Y_C \text{ for } i = 1, \ldots, 12, \\ Y_{12+i} &\sim Y_H \text{ for } i = 1, \ldots, 22, \quad \text{and} \\ Y_{34+i} &\sim Y_O \text{ for } i = 1, \ldots, 11. \end{aligned}$$

nominal mass	342	343 (+1)	344 (+2)	345 (+3)	346 (+4)
abundance %	85.3428	11.6595	2.6656	0.2913	0.0371
mean peak mass	342.116215	343.119653	344.121190	345.124158	346.125990

Table 7.4: Isotope pattern (isotope distribution and mean peak masses) of sucrose $C_{12}H_{22}O_{11}$. Peaks with nominal mass $347, \ldots, 398$ have abundances of less than 0.01 %.

In an ideal mass spectrum, normalized peak intensities correspond to the isotope distribution of the molecule. For ease of exposition, the peak at monoisotopic mass is also called *monoisotopic*, the following peaks are referred to as +1, +2, ...peaks. (In the literature, these peaks are also called $A+1, A+2, \ldots$, to distinguish them from charges; but we do not consider charges here.) The number of non-zero entries in the isotope distribution of a molecule is

$$\text{number of non-zero entries} = i_C + i_H + i_N + 2i_O + 4i_S + 1 \tag{7.2}$$

where again, i_E denotes the multiplicity of element E in the molecule, $E \in \{C, H, N, O, P, S\}$. (Strictly speaking, this holds if the molecule contains at least one atom of C, H, N, or O; for a molecule consisting only of sulfur and phosphorus, we have to replace "=" by "≤".) Clearly, this is much less than the number of isotopologues, see (7.1): For example, sucrose $C_{12}H_{22}O_{11}$ has $12 + 22 + 2 \cdot 11 + 1 = 57$ non-zero entries, ranging from nominal mass 342 to 398. See Table 7.3 for the isotope distribution of sucrose. Put differently, if Y is the random variable of sucrose, then $\mathbb{P}(Y = 342) = 0.8534$. Peak intensities quickly deteriorate for increasing nominal mass, and $\mathbb{P}(Y \geq 347) < 0.00004$.

A perfect mass spectrometry instrument would show individual isotopologue peaks, but the "imperfection of mass spectrometry" results in +1, +2, ... isotope peaks that, in fact, are superpositions of peaks with almost identical mass. We have introduced above a model for the intensity of the superimposed peak; but what about its mass? It is reasonable to assume that the mass of a peak in the isotope pattern, is the mean mass of all isotopologues that add to its intensity. We now formalize this idea: For each element $E \in \Sigma$ we define another random variable X_E, representing the mass of the natural isotopes. Random variables X_E and Y_E are correlated: In fact, X_E can be viewed as a function λ of Y_E and E, $X_E = \lambda_E(Y_E)$. For example, X_C with state space $\{12, 13.003355\}$ and

$$\mathbb{P}\big(X_C = 12\big) = 0.98930, \quad \mathbb{P}\big(X_C = 13.003355\big) = 0.01070$$

is the random variable of carbon, and we have $X_C = 12$ if and only if $Y_C = 12$. Given a molecule consisting of l atoms, we assign to the ith atom, $i = 1, \ldots, l$, a random variable X_i such that $X_i \sim X_{E_i}$, where E_i is the element of the ith atom. Now we can represent the molecule's *mass distribution* by the random variable $X := X_1 + \ldots + X_l$. Clearly, X and Y are correlated, where Y is the isotope distribution of the molecule.

For mass distribution $X = X_1 + \ldots + X_l$ and isotope distribution $Y = Y_1 + \cdots + Y_l$ of a molecule with elements $E_1, \ldots, E_l$ and monoisotopic nominal mass N, the mean peak mass m_n of the $+n$ peak can be calculated as:

$$\begin{aligned} m_n &= \mathbb{E}\big(X \;\big|\; Y = N + n\big) \\ &= \sum_{\sum_i N_i = N+n} \frac{\mathbb{P}(Y_1 = N_1, \ldots, Y_l = N_l)}{\mathbb{P}(Y = N + n)} \Big(\lambda(N_1, E_1) + \cdots + \lambda(N_l, E_l)\Big) \end{aligned} \tag{7.3}$$

where the sum is taken over all vectors $\vec{N} = (N_1, \ldots, N_l) \in \mathbb{N}^l$ satisfying $\sum_i N_i = N + n$. We refer to the isotope distribution together with the mean peak masses as the molecule's *isotope pattern*. See Table 7.4 for the isotope pattern of sucrose.

7.3 Simulating isotope patterns

In the following, we will "separate" isotope patterns from the monoisotopic nominal mass of the molecule: If two molecular formulas differ by a single phosphorus atom, then the resulting isotope patterns are

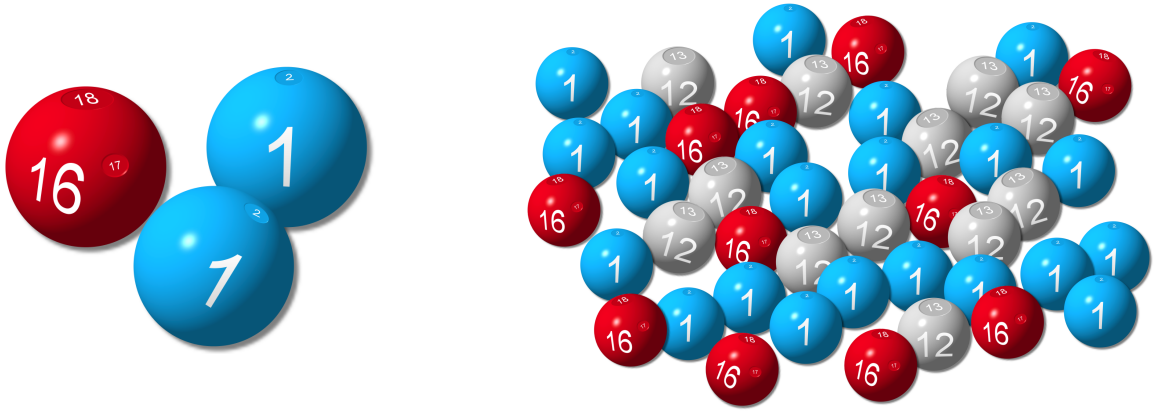


Figure 7.1: Throwing dice to simulate an isotope distribution, for water H_2O (left) and sucrose $C_{12}H_{22}O_{11}$ (right). H-dice have two faces, where a '1' is rolled in 99.988 % of the cases, and a '2' in 0.012 %. C-dice and O-dice are made analogously, but O-dice have three faces. The rolled number is found on the face that touches the table, compare with four-sided dice (tetrahedra, D4) used in role-playing games.

identical, only shifted by the mass of a single phosphorus atom. In other words: It is of no interest for the isotope pattern what the actual nominal mass N of the molecule is. So, we write nominal masses of isotopes as $N+n$, corresponding to the $+n$ peak of the isotope pattern. The monoisotopic peak will also be referred to as "the first non-zero value of the distribution" because obviously, no isotope can have smaller mass.

We start with the computation of the isotope distribution, as this will be needed to compute mean peak masses. Let us compute the isotope distribution of sucrose $C_{12}H_{22}O_{11}$ by hand, see Table 7.3. To do so, we put 100 000 marbles in a bag: 99 988 will be marked "1" and 12 marbles will be marked "2". We label the bag with an 'H'. We prepare a second bag labeled 'C' that contains 9 893 marbles marked "12" and 107 marbles marked "13". In a third bag labeled 'O' we put 99 757 marbles marked "16", 38 marbles marked "17", and 205 marbles marked "18". At random, pull a marble from the bag labeled 'C', write down the number, put it back. Repeat 11 more times. Do the same for bag 'H' with 22 repetitions, and for the bag 'O' with 11 repetitions. Sum up all numbers, record the sum on a second piece of paper. Repeat 10 000 times, count how often you have computed each sum — voilà, you have just simulated an isotope distribution. Another way to do this is throwing dice, see Fig. 7.1. Obviously, these two methods are not very efficient[3] to simulate isotope distributions, neither by hand nor on a computer: Doing so is not only time consuming but, even worse, the simulated isotope distribution can still differ significantly from the distribution you would get for an infinite number of repetitions. Can we do better?

Computing the complete isotope distribution is somewhat tedious, as there are many intensities that we compute in vain, see again Table 7.3 where peaks at nominal masses 347 to 398 will not be detectable in any mass spectrometer. In fact, isotope distributions decrease rapidly for all molecules over the alphabet of elements CHNOPS. To further substantiate this claim empirically, we extracted all molecular formulas from the KEGG COMPOUND database [171] (release 42.0) that have elements CHNOPS and mass below 3000 Da. Among the resulting 11479 molecular formulas, not a single entry has an intensity of the +10 peak larger than 0.007 %. Clearly, the corresponding peak must be lost in the noise of the experimental mass spectrum. We consider the worst case of a "sulfur-only" molecule in Exercise 7.2.

The above implies that we can restrict our computations to the first $n_{\max}$ non-zero values of the distribution, where $n_{\max}$ is a small constant such as $n_{\max} = 10$. In the following, these $n_{\max}$ values will be referred to (slightly imprecisely) as the *isotope distribution*.

[3] Doing these naïve simulations can nevertheless teach you something about the variations of peak intensities you have to expect in practice, as you are looking at a finite number N of molecules that generated the measured isotope distribution in the instrument; see also Exercise 7.8.

We begin with "pure" molecular formulas made from a single element, such as H_{63}. Clearly, such molecular formulas are "unreasonable" as they usually do not correspond to a molecule. But that should not stop us from calculating the corresponding isotope distribution!

The elements hydrogen, carbon, and nitrogen have only two natural isotopes. Thus, the isotope distribution of a molecule E_l consisting of l atoms of element E with $E \in \{\mathrm{H}, \mathrm{C}, \mathrm{N}\}$, follows a binomial distribution: Let q_n denote the probability that E_l has nominal mass $N + n$, where N is the monoisotopic nominal mass of E_l. Then,

$$q_n = \binom{l}{n} p^{l-n}(1-p)^n, \tag{7.4}$$

where p is the relative abundance of the monoisotopic isotope of element E. The values of the q_n can be computed iteratively, since $q_0 = p^l$ and

$$q_{n+1} = \frac{l-n}{n+1} \cdot \frac{1-p}{p} q_n \quad \text{for } n \geq 0, \tag{7.5}$$

thus the total computation time is $O(n_{\max})$.

Where an element E has $r > 2$ isotopes (such as oxygen and sulfur), the isotope distribution of E_l can *in theory* be computed as follows: Let p_i for $i = 0, \ldots, r-1$ denote the probability of occurrence of the ith isotope. (For sulfur, we treat the non-existing isotope $^{35}\mathrm{S}$ as an isotope with abundance zero, so that the ith isotope always has nominal mass $+i$.) Then, the probability that E_l has nominal mass $N + n$ is

$$q_n := \mathbb{P}(E_l \text{ has nominal mass } N+n\,) = \sum \binom{l}{l_0, l_1, \ldots, l_{r-1}} \cdot \prod_{i=0}^{r-1} p_i^{l_i}, \tag{7.6}$$

where the sum runs over all $l_0, \ldots, l_{r-1} \geq 0$ satisfying $\sum_{i=0}^{r-1} l_i = l$ and $\sum_{i=0}^{r-1} i \cdot l_i = n$ [155]. The tuples $(l_0, \ldots, l_{r-1})$ satisfying $\sum i \cdot l_i = n$ are the integer partitions of n into parts of size at most $r-1$. To compute all partitions, a greedy algorithm with a simple recursion can be employed. However, this approach faces the problem that the number of summands in (7.6) grows rapidly, as a polynomial in n of degree $r-2$ [352], and is impractical in practice.

We now present a smarter way to compute the isotope distribution of O_l and S_l. Let Y and Y' be two discrete random variables with state spaces $\Omega, \Omega' \subseteq \mathbb{N}$. Recall that we can compute the distribution of the random variable $Z := Y + Y'$ as

$$\mathbb{P}(Z = x) = \sum_y \mathbb{P}(Y = x - y) \cdot \mathbb{P}(Y' = y), \tag{7.7}$$

see (5.3) in Sec. 5.4. If we restrict ourselves to the first $n_{\max}$ non-zero values of this distribution, we can compute it in $O(n_{\max}^2)$ time. We briefly recall the details: Let $P_Y[0 \ldots n_{\max} - 1]$ and $P_{Y'}[0 \ldots n_{\max} - 1]$ be the first $n_{\max}$ non-zero values of the distributions of Y and Y'. Then,

$$P_Y[n] = \mathbb{P}(Y = N + n) \quad \text{and} \quad P_{Y'}[n] = \mathbb{P}(Y' = N' + n)$$

holds for $n = 0, \ldots, n_{\max} - 1$ and some $N, N' \in \mathbb{N}$; furthermore, $P_Y[0] > 0$ and $P_{Y'}[0] > 0$, as well as $\mathbb{P}(Y = n) = 0$ for $n < N$ and $\mathbb{P}(Y' = n) = 0$ for $n < N'$. We compute an array $P_Z[0 \ldots n_{\max} - 1]$ as

$$P_Z[n] \leftarrow \sum_{i=0}^{n} P_Y[i] \cdot P_{Y'}[n-i] \quad \text{for } n = 0, \ldots, n_{\max} - 1 \tag{7.8}$$

and find that $P_Z[n] = \mathbb{P}(Z = N + N' + n)$ for all $n = 0, \ldots, n_{\max} - 1$, as well as $\mathbb{P}(Z = n) = 0$ for all $n < N + N'$, where $Z = Y + Y'$. We will see below that (7.8) also allows us to swiftly compute the isotope distribution of an arbitrary molecular formula.

We can compute the isotope distributions of oxygen O_l and sulfur S_l by iterative convolution: For example, the isotope distribution of O_l is computed by l times convolving the distribution of oxygen. This

```
1:  function SmartRussian(isotope distribution P, integer l)
2:      isotope distributions Q = Q[0 ... n_max − 1] and Q' = Q'[0 ... n_max − 1]
3:      Q[0] ← 1, Q[i] ← 0 for i = 1, ..., n_max − 1
4:      Q' ← P
5:      while l > 0 do
6:          if l is odd then
7:              Convolve Q and Q', store result in Q
8:          end if
9:          Convolve Q' and Q', store result in Q'
10:         l ← ⌊l/2⌋
11:     end while
12:     return isotope distribution Q
13: end function
```

Algorithm 7.1: Smart Russian algorithm for computing the isotope distribution of O_l and S_l, as well as other elements with three or more natural isotopes.

results in $O(l n_{\max}^2)$ time for computing the first $n_{\max}$ coefficients of the distribution of O_l and S_l. Actually, we can do better than that: Russian multiplication[4] allows us to compute the product $a \cdot b$ of two integers by repeatedly doubling one, halving the other: For example,

$$\begin{aligned} 133 \cdot 177 &= 133 \cdot 2^0 + 133 \cdot 2^4 + 133 \cdot 2^5 + 133 \cdot 2^7 \\ &= 133 + 2128 + 4256 + 17024 = 23541 \end{aligned}$$

as $177 = 1 + 16 + 32 + 128 = 2^0 + 2^4 + 2^5 + 2^7$. This also works for computing a^b:

$$133^{177} = 133^{2^0} \cdot 133^{2^4} \cdot 133^{2^5} \cdot 133^{2^7} \approx 8.35 \cdot 10^{375}$$

Similarly, we can compute the distribution of the random variable $Z = Z_1 + \cdots + Z_l$ where $Z_i \sim Z_1$, see Algorithm 7.1. Limiting ourselves to the first $n_{\max}$ coefficients of the distribution, this results in running time $O(n_{\max}^2 \log l)$.

But although Algorithm 7.1 is quite fast, we can do better, using a simple trick: We shift these computations into the preprocessing phase, storing results in memory. For that, we have to choose some fixed L, and store isotope distributions for O_l and S_l where $l = 1, \ldots, L$. This results in $O(n_{\max} L)$ memory for every element. Note that L is small in practice: For example, 256 oxygen atoms already have a mass of about 4096 Da, most likely exceeding the relevant mass range.

Assume you want to compute the isotope distribution of some molecular formula $E_{L'}$ but have only stored the isotope distributions of molecular formulas E_l for $l = 1, \ldots, L$, where $L < L'$. In this case, we rely as much as possible on what we have previously computed, and use a modified version of the Russian folding algorithm for the rest. The somewhat obvious Algorithm 7.3 can be found at the end of this chapter.

Now, the algorithm for computing the actual isotope pattern of an arbitrary molecular formula, becomes rather trivial, see Algorithm 7.2: For molecules consisting of different elements, we first compute or look up the isotope distributions of the individual elements. Then, we combine these distributions by convolution in $O(|\Sigma| \cdot n_{\max}^2)$ time. There, we assume that isotope distributions have been precomputed for all elements $E \in$ CHNOS. Alternatively, these distributions can be computed on the fly for $E \in \{C, H, N\}$ using (7.5). Also, we can use Algorithm 7.3 instead of directly accessing the precomputed distributions; we refrained from doing so solely for readability. Case closed.

We now come to the more challenging problem of efficiently computing the mean peak masses of a distribution. Doing so using the definition $m_n = \mathbb{E}(X \mid Y = N + n)$ is highly inefficient, because we

[4] Also known as smart Russian multiplication, Russian peasant multiplication, ancient Egyptian multiplication, or Egyptian multiplication.

1: **function** IsotopeDistribution(molecular formula $C_{i_C}H_{i_H}N_{i_N}O_{i_O}P_{i_P}S_{i_S}$)
2: distribution $Q := P_H[i_H]$
3: Fold Q and $P_C[i_C]$, store result in Q
4: Fold Q and $P_N[i_N]$, store result in Q
5: Fold Q and $P_O[i_O]$, store result in Q
6: Fold Q and $P_S[i_S]$, store result in Q
7: return isotope distribution Q
8: **end function**

Algorithm 7.2: Compute the isotope distribution of an arbitrary molecular formula with i_E atoms of element E, over the alphabet CHNOPS of elements. We assume that isotope distributions $P_E[i]$ for molecular formulas E_l for all CHNOS have been precomputed.

have to sum up over all isotopologues. Pruning strategies have been developed to speed up computation [372], but pruning leads to a loss of accuracy [297]. We now present a simple recurrence for computing these masses analogous to the convolution of distributions: Let $Y = Y_1 + \cdots + Y_l$ and $Y' = Y'_1 + \cdots + Y'_L$ be isotope distributions of two molecules with monoisotopic nominal masses N and N', respectively. Let $p_n := \mathbb{P}(Y = N + n)$ and $q_n := \mathbb{P}(Y' = N' + n)$ denote the corresponding probabilities, m_n and m'_n the mean peak masses of the $+n$ peaks. Consider the random variable $Z = Y + Y'$ with monoisotopic nominal mass $N + N'$.

Theorem 2. *The mean peak mass M_n of the $+n$ peak of the isotope pattern for random variable $Z = Y + Y'$ can be computed as:*

$$M_n = \frac{1}{\sum_{j=0}^{n} p_j q_{n-j}} \cdot \sum_{j=0}^{n} p_j q_{n-j} \left(m_j + m'_{n-j} \right). \tag{7.9}$$

The mean peak masses M_n must not be mixed up with the precursor mass M from Chapter 2. Note that $\sum_{j=0}^{n} p_j q_{n-j} = \mathbb{P}(Z = N + N' + n)$. Since by independence, $\mathbb{P}(Y_1 = N_1, \ldots, Y_l = N_l) = \prod_i \mathbb{P}(Y_i = N_i)$, the theorem follows by rearranging summands. A formal proof can be found in Sec. 7.8.

The theorem allows us to "convolve" mean peak masses of two distributions to compute the mean peak masses of their sum. This implies that we can compute mean peak masses as efficiently as the distribution itself.

7.4 Faster approximation by the fast Fourier transform

The following is slightly more mathematical than the rest of this chapter. (I hope that I got all the math details right; I am not an expert in Fourier theory; Claude helped. 😉) What we want to do, is to further speed up computations of the previous section for simulating an isotope pattern. It turns out that this is indeed possible, with some elegant mathematical theory. Unfortunately, the impact in practice is rather limited, resulting in a speedup of two- to threefold for small molecules and peptides.

The convolution theorem tells us that a convolution of functions in the time domain corresponds to a multiplication of functions in the frequency domain, and vice versa: Formally, given integrable functions f, g with Fourier transforms $\hat{f}, \hat{g}$, then the convolution

$$h(x) = \int_{-\infty}^{\infty} f(y) g(x - y)\, dy$$

has Fourier transform

$$\hat{h}(\xi) = \hat{f}(\xi) \cdot \hat{g}(\xi).$$

Formally, if $\mathcal{F}$ denotes the Fourier transform operator, then

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

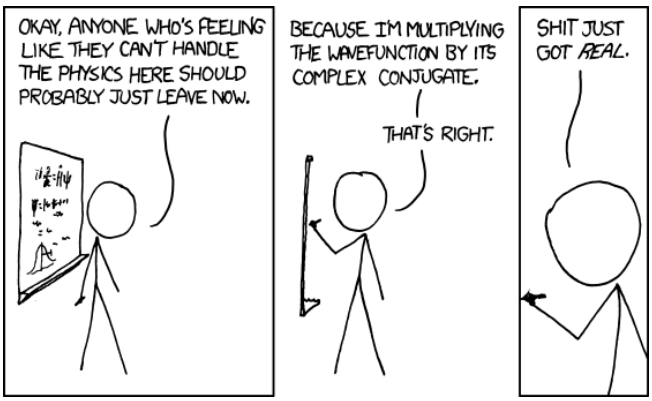


Figure 7.2: Complex conjugates by xkcd (https://xkcd.com/849/). Remember, if $x = a + b \cdot i$ then $x \cdot x^* = a^2 + b^2$ where $a, b \in \mathbb{R}$.

where "$*$" denotes the convolution operator and "$\cdot$" denotes the pointwise multiplication. There exist four variants of the Fourier transform, where time and frequency domain can be either discrete or continuous.

For us, the interesting variant is the discrete Fourier transform, where both time and frequency domain are discretized: We transform a sequence of N complex numbers $x_0, \ldots, x_{N-1}$ into another sequence of complex numbers $X_0, \ldots, X_{N-1}$ through

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \exp\left(-\tfrac{2\pi i}{N} kn\right) = \sum_{n=0}^{N-1} x_n \cdot \Big(\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)\Big).$$

We can think of the x_n as samples of the underlying periodic function f; the discrete-time Fourier transform of such a function is periodic, and the X_n are samples of one cycle. In agreement with the usual notation of the field, N denotes the length of the transform in this section, not a nominal mass.

Note that both x_n and X_n are complex; but if the x_n are real-valued, as will be the case here, then $X_k = X^*_{N-k}$ where $x^* = a - b \cdot i$ is the *complex conjugate* of a complex number $x = a + b \cdot i$, $a, b \in \mathbb{R}$; see Fig. 7.2. This means that we have to store only the complex numbers $X_0, \ldots, X_{N/2}$ (assuming for simplicity that N is even), so the discrete Fourier transform of N real-valued numbers is again N real-valued numbers. Recall that the product of two complex numbers $a+i\cdot b$ and $c+i\cdot d$ for $a, b, c, d \in \mathbb{R}$ is simply $(ac-bd)+i\cdot(ad+bc)$. Here, $i = \sqrt{-1}$ is the imaginary unit.

Now, the important point is that the convolution theorem still holds for discrete time and frequency domain: If $X_0, \ldots, X_{N-1}$ are the discrete Fourier transform of $x_0, \ldots, x_{N-1}$ and $Y_0, \ldots, Y_{N-1}$ are the discrete Fourier transform of $y_0, \ldots, y_{N-1}$, then

$$Z_k = X_k \cdot Y_k \quad \text{for } k = 0, \ldots, N-1$$

is the discrete Fourier transform of $z_0, \ldots, z_{N-1}$ with

$$z_n = \sum_{j=0}^{N-1} x_j \cdot y_{(n-j) \bmod N}.$$

Note the "$\bmod N$": The discrete convolution is *cyclic*, so indices that run over the end of the sequence start again at its beginning; we will come back to this below. To get back from the frequency domain, we use the inverse discrete Fourier transform

Claude helped!

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot \exp\left(\tfrac{2\pi i}{N} kn\right),$$

which differs from the transform only in the sign of the exponent and the factor $\frac{1}{N}$. To simulate the isotope distribution of a molecular formula with i_E atoms of element E, we pad the isotope distribution of a single

atom of each element with zeros to length N, and compute its discrete Fourier transform. For each k, we raise the k-th value of element E to the power i_E, and multiply over all elements. Finally, a single inverse transform gives us the isotope distribution.

Computing the discrete Fourier transform from its definition requires $O(N^2)$ time. The Fast Fourier Transform (FFT) [63] does the same in $O(N \log N)$ time, using divide and conquer: For N a power of two, the transform of length N is computed from two transforms of length $\frac{N}{2}$, one for the even and one for the odd indices. The same holds for the inverse transform.

The nice point is that we do the same for expected masses of the isotope pattern: Using the notation of Theorem 2 and $r_n := \sum_{j=0}^{n} p_j q_{n-j}$, we can rewrite (7.9) as

$$r_n M_n = \sum_{j=0}^{n} (p_j m_j) \cdot q_{n-j} + \sum_{j=0}^{n} p_j \cdot (q_{n-j} m'_{n-j}).$$

So, the "mass-weighted" distribution $(r_n M_n)_n$ is the sum of two convolutions, and the convolution theorem tells us that

$$\mathcal{F}\{rM\} = \mathcal{F}\{pm\} \cdot \mathcal{F}\{q\} + \mathcal{F}\{p\} \cdot \mathcal{F}\{qm'\}.$$

This looks suspiciously like the product rule $(fg)' = f'g + fg'$ for derivatives, and it behaves like it: By induction, the transforms for a molecular formula E_l are $\mathcal{F}\{p\}^l$ for the isotope distribution and $l \cdot \mathcal{F}\{pm\} \cdot \mathcal{F}\{p\}^{l-1}$ for the mass-weighted distribution, just like the derivative of x^l. Here, p and pm are the isotope distribution and mass-weighted distribution of a single atom of E, that is, isotope abundances and isotope abundances times isotope masses. At the end, we apply two inverse transforms and divide, $M_n = (rM)_n / r_n$. For peaks of tiny intensity, this division amplifies rounding errors; but we are not interested in these peaks, anyway. Fernandez-de-Cossio Diaz and Fernandez-de-Cossio [111] also compute mean peak masses via the Fourier transform.

Be warned that implementing the FFT can easily result in numerical instabilities; you should rely on an existing implementation that avoids such instabilities.

Unfortunately, the above computations are only an *approximation* of the true numbers: Above, we have — explicitly or implicitly — assumed that the signal (the isotope distribution) is periodic, which is not the case. We have done so because the Fourier transform of a non-periodic discrete signal is continuous, and we do not want to multiply continuous functions in our computations for sure. This is a common trick also used, say, in image compression, and results in block boundary artifacts if you use high compression rates. Similarly, the discrete Fourier transform will "carry over" all parts of the isotope pattern which require indices N and beyond: This part of the isotope pattern cannot "get lost" as it should be, but rather increases, say, the monoisotopic peak. Therefore, N must be chosen sufficiently large, and in particular larger than $n_{\max}$, which is the number of isotope peaks we are interested in. Since the isotope distribution of any molecular formula is quickly vanishing, this is usually not a problem in practice; in fact, you can push N so that other sources of numerical errors take over, see Sec. 15.3. Remember that *all* computations carried out in a computer, are only an approximation of the real thing…

Was it all worth it? For simplicity, let us concentrate on the simulation of isotope distributions, without expected peak masses. On the theoretical side, we have done a wonderful job: Instead of running time $O(l n_{\max}^2)$ for convolving the isotope distributions of the $l = |\Sigma|$ elements, we now have running time $O(l n_{\max} + n_{\max} \log n_{\max})$ where $l n_{\max}$ is for the convolution part and $n_{\max} \log n_{\max}$ corresponds to the FFT. Note that $x^j = \exp(j \log(x))$ can be computed in $O(1)$ time, and that we assume that the Fourier transforms of each element's isotope distribution were computed and stored during preprocessing. With that, we only need a single FFT at the end of our computations.

In practice, computing and storing Fourier transforms of each element's isotope distribution is necessary for all elements including hydrogen and carbon, as we cannot use the "binomial distribution trick" from (7.5). But then, we still have to compute x^j where j is the number of atoms of this element; to avoid the $\Theta(\log j)$ multiplications required for this computation, we instead store the FFTs for each element *and each multiplicity*, up to some upper limit. (Using equality $x^j = \exp(j \log(x))$ is *worse in practice* for

reasonable j, as it requires calling the time-expensive exponential and logarithm functions.) Doing so, we again need just a single FFT at the end of our computations.

Now, was it worth the effort in practice? We first consider small molecules and peptides; let us consider a reasonable $n_{\max}$ such as $n_{\max} = 8$, so we are interested in eight values of the isotope distribution. Each convolution of isotope distributions requires $\frac{8\cdot 9}{2} = 36$ multiplications using the naïve approach. In comparison, we now have to multiply four complex numbers (recall $X_k = X^*_{N-k}$) which corresponds to 16 real-valued multiplications. Counting both multiplications and additions, which cost about the same on modern processors, the final inverse FFT costs about as much as a single direct convolution; for a molecular formula over CHNOS, we have still roughly doubled the speed of the algorithm (256 vs. 144 operations). For 16 peaks, the advantage grows to roughly threefold (1024 vs. 344 operations), as the direct convolution needs quadratic time. This comes at the price that we should choose N slightly larger than $n_{\max}$, and that our computations are only an approximation, though probably a highly accurate one. If simulating isotope patterns is the time-critical step in your analysis pipeline, you will indeed see running time improvements of up to a factor of 2 in practice. Seen the other way around, you can avoid the Fourier transform overhead[5] at the cost of roughly doubled running times.

Claude helped!

What about proteins? Here, the FFT approach can result in massive improvements in running time, as we are substituting a linear term by its logarithm. Unfortunately, simulating the isotope pattern of a large protein — where the speedup will be most dramatic — is an intellectual exercise with basically no practical relevance, as we will see in Sec. 7.6. Hence, it does not seem worthwhile to simulate the isotope pattern of a protein beyond 20 000, maybe 30 000 Dalton. For proteins below this threshold, it will depend on your application how much you can speed up computations by switching to the frequency domain.

7.5 Sulfur and other mavericks

What is so special about sulfur, that we have to treat it differently than the other elements? First, look at mass differences of isotopes: The mass difference $\mu(^{13}\mathrm{C}) - \mu(^{12}\mathrm{C}) = 1.003355$ is larger than one, so the isotope peaks of a carbon molecule are farther to the "right" than nominal masses suggest. In contrast, $\mu(^{34}\mathrm{S}) - \mu(^{32}\mathrm{S}) = 1.995796$, so isotope peaks of a sulfur molecule are farther to the "left" than nominal masses suggest. But nitrogen and even hydrogen also show strong deviations in the mass difference of isotopes, and we do not treat them separately. So, what is special about sulfur?

The answer is somewhat more subtle: Our assumption that an isotope peak is a superposition of all isotopologues with identical nominal mass, only holds if mass differences between subsequent isotopologues are small, or if intensities of outlier isotopologues are very small.

See Table 7.2 for the isotopologues of sucrose: There are seven isotopologues with nominal mass 344, ranging in mass from 344.120460 to 344.128769, an interval of 0.008309 Da width. But the mass difference between any two subsequent isotopologues is much smaller, namely 0.002465 at maximum. This mass difference is about 7 ppm, and even though resolving peaks is a matter of resolution and not of mass accuracy, it should be easy to believe that such peaks can easily "smear" into a single peak in a mass spectrum.

Now, let us do the gedankenexperiment that the molecular formula contains an additional sulfur with nominal mass 32 — what are the resulting isotopologues with nominal mass 344 + 32 = 376? The isotopologues that use the ^{32}S sulfur isotope, are the same as those displayed in Table 7.2, only shifted by 31.972071, and range from 376.092531 to 376.100840. The ^{33}S isotope of sulfur will result in several additional isotopologues of low intensity, which we may ignore. But the ^{34}S isotope of sulfur results in a *single* isotopologue with mass 376.084082, at distance 0.008449 Da to the closest isotopologue.

What this means in practice is that we can sometimes observe two +2 peaks, where one corresponds to the isotopologue with a single ^{34}S, and the other one corresponds to all other isotopologues with this

[5] One may argue that using the Fourier transform makes the code for simulating isotope patterns simpler and easier to read; personally, I do not think that directly computing a convolution via (7.8) and (7.9) is particularly complicated, either. Be reminded that you should *not* try to implement the fast Fourier transform yourself.

nominal mass. If the resolution of the instrument gets better, you might observe even more peaks in the experimental isotope pattern.

There are two ways to deal with this anomaly in practice: First, we can "calculate it away". Consider the sample spectrum: If m, m' are peak masses and h, h' intensities with identical nominal mass, we can replace them by a single peak with mass $(hm + h'm')/(h + h')$ and intensity $h + h'$. This approach has the advantage that it works out of the box; it has the disadvantage that we are sacrificing information. I would argue that the amount of lost information is rather small: We will be able to identify molecular formulas with sulfur also from the shifted +2 peak; but beyond that, not much more can be learned from the split peak.

Second, we can integrate the instrument parameter "resolution" into our calculations. From a computational perspective, this is possible without increasing running times too much, see Exercise 7.12. But on a conceptual level, it is somewhat more complicated which peaks we merge and which peaks we do not: Assume that peaks a and b have mass difference such that the resolution parameter tells us to merge these two; assume further that peaks b and c have such mass difference, too. Now, if we first merge peaks a and b, the resulting peak a+b may have mass difference to c such that our resolution parameter tells us *not* to merge a+b and c. In contrast, if we start merging b+c, we might not merge a and b+c. I do not know if there is a generally applicable procedure which produces an unambiguous result, allows for a swift implementation *and* produces substantially better results, to justify the overhead. You may instead directly choose to simulate isotopologue patterns, see Sec. 7.7.

7.6 Isotope patterns of peptides and proteins

The obvious way to compute the isotope pattern of a peptide, is to first compute its molecular formula, then to simulate the isotope pattern using the methods from Sec. 7.3. An alternative approach is to precompute isotope patterns for each amino acid (with multiplicities), then to fold these isotope patterns according to their frequency in the peptide. But this is advantageous only if the peptide contains at most five different amino acids; otherwise, the detour over the molecular formula is faster.

Unlike metabolites, there is less information we can gain from the experimental isotope pattern of a peptide: Isotope patterns of sufficiently large peptides often look mostly indistinguishable, see below for a detailed discussion. Consequently, some authors have defined the *averagine* amino acid, for example $C_{4.9384}H_{7.7583}N_{1.3577}O_{1.4773}S_{0.0417}$ from [320]. This averagine has monoisotopic mass 111.054306; if you have observed an unknown peptide with monoisotopic mass m, you can estimate that an average peptide has 4.9384α carbon, 7.7583α hydrogen, 1.3577α nitrogen, 1.4773α oxygen and 0.0417α sulfur, where $\alpha = m/111.054306$. You first choose CNOS, round appropriately and "fill up" the molecular formula with hydrogen so that at least the nominal mass of the peptide fits. Alternatively, you can use the fast Fourier transform from Sec. 7.4 which allows non-integer numbers of atoms; but the results will probably be mostly indistinguishable.

From the above, it is understood that the isotope pattern does not allow us to determine the number of amino acids *de novo*, see also Sec. 8.6. But simulating isotope patterns of peptides may nevertheless be useful: For example, when doing database search we have an extremely restricted set of candidates — thousands or tens of thousands of candidates — and many of these candidates will have a theoretical isotope pattern which is very different from the experimental one. Here, integrating the isotope pattern similarity into the overall candidate score may help us to rule out certain candidates which would not be possible using tandem MS data alone. Furthermore, one piece of information may be extracted from the isotope pattern without the need for a database lookup: Sulfur atoms will usually result in notably different isotope patterns for peptides, see Sec. 7.5. Thus, one may determine the combined number of cysteine/methionine residues from the isotope pattern of a peptide. Similarly, isotope patterns will come in handy when you are analyzing data from isotopic labeling experiments: You may, for example, grow bacteria on "all ^{13}C" medium. If some molecule is made from carbon which in part follows the natural isotope distribution and in part is "all ^{13}C", it will clearly affect its isotope pattern; hence, you can

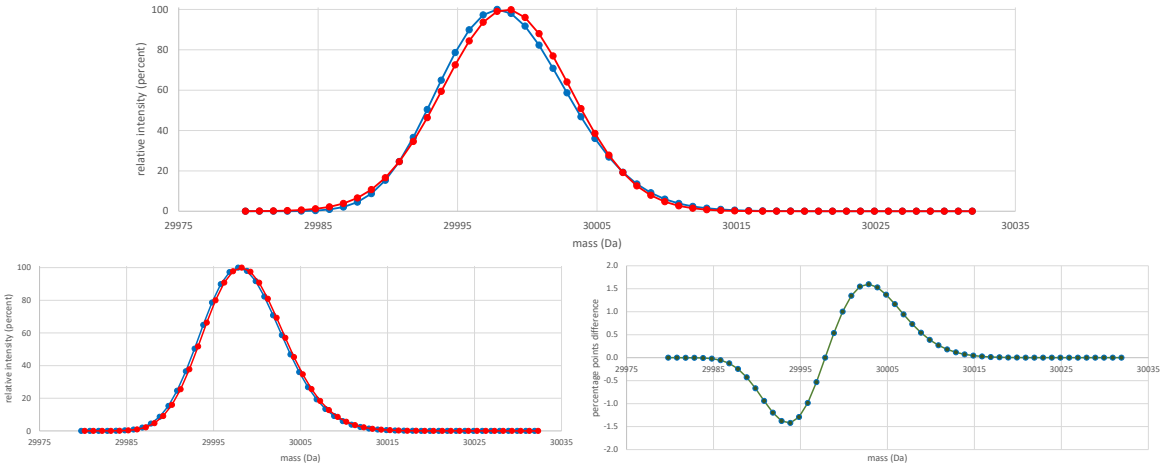


Figure 7.3: Top: Simulated isotope pattern of Galectin-3 of the green anole lizard $C_{1343}H_{2039}N_{367}O_{397}S_{10}$ (blue) vs. normal distribution with $\mu = 29998.491$ and $\sigma^2 = 20.894996$ (red), normalized to maximum intensity 100 %. Lines included for better readability; this is *not a mass spectrum* where, on a high-resolution instrument, separate peaks will be recorded. The isotope distribution is slightly skewed but otherwise, the fit is of high quality. Bottom, left: Simulated isotope patterns of Galectin-3 and the human "Microtubule-associated protein RP/EB family member 1" $C_{1340}H_{2099}N_{355}O_{408}S_9$. Bottom, right: Differences (in percentage points) of the two normalized protein isotope distributions, using rounded masses.

recover the fractions of labeled carbons from there, see also Exercise 7.11. I assume that numerous other applications exist where isotope patterns can give useful information about peptides.

For proteins up to, say, 30 000 Da, simulating isotope patterns can also be helpful, as long as you have a reasonably small number of hypotheses about the protein identity. (An extreme case with substantial changes in the isotope pattern is that of hemoproteins measured with or without the iron ion in the coordination complex; clearly, this also changes the mass of the protein.) But for larger proteins with identical molecular weight, the central limit theorem will make sure that their isotope *distributions* look practically indistinguishable: This theorem states that the sum of independent random variables tends toward a normal distribution (certain restrictions apply). Do not get me wrong here: I *do not* propose to use the normal distribution to simulate the isotope pattern; what I am saying is that for large proteins, the resulting isotope distribution will look "mostly normal" and, hence, does not contain much information about the identity of the protein. See Fig. 7.3. Clearly, we do not have to simulate the isotope distribution to estimate its central moments: In detail, the variance of independent random variables is just the sum of variances of the random variables, and the same holds true for the third central moment.

In reality, the situation is even worse: With the exception of cysteine/methionine, amino acid residues have "rather similar" molecular formulas, see the averagine introduced above. Two randomly chosen proteins with similar molecular weight will therefore also have similar molecular formulas: For example, the randomly picked proteins[6] Galectin-3 of the green anole lizard and the human "Microtubule-associated protein RP/EB family member 1" have molecular formulas $C_{1343}H_{2039}N_{367}O_{397}S_{10}$ and $C_{1340}H_{2099}N_{355}O_{408}S_9$, respectively; the intersection of these molecular formulas is $C_{1340}H_{2039}N_{355}O_{397}S_9$ whereas the molecular formula difference is a mere $C_3H_{-60}N_{12}O_{-11}S$. Hence, the resulting isotope distributions are more similar to one another than to the normal distribution, see again Fig. 7.3. Notably, the monoisotopic masses of the two proteins *are* very different (29979.785 Da vs. 29980.190 Da, 13 ppm mass difference) but the same is true for the molecular weight (29998.492 Da vs. 29998.784 Da, 10 ppm weight difference). Consequently, the isotope pattern will not give you substantially more information

[6] Taken from UniProt, https://www.uniprot.org/, query "mass [sic] between 29999 and 30001 Da"; UniProt uses standard atomic weights.

than the monoisotopic mass. As always, these statements hold *in general*: It is easy to come up with examples where the isotope patterns of two large proteins *are* easily distinguishable, such as one protein with many cysteine/methionine and one protein without; see the "sweet spot" discussion of Sec. 8.6 below.

Unfortunately, there is yet another issue when simulating isotope patterns of large proteins: Whereas masses of isotopes are physical constants, we noted above that natural abundances of these isotopes are *not*. They vary not only if we go to other planets or moons; even on Earth, differences can be substantial, for example between marine and land organisms and, in particular, for hydrogen. In fact, isotope abundances can differ between two compound classes (sugars, lipids) in the same organism. This has no dramatic effect on the isotope patterns of small molecules or peptides, but I would argue that it renders a simulated isotope pattern of a large protein mostly useless; in fact, both the molecular weight and the mode of the isotope distribution can shift by many Dalton if your protein gets large enough. For this reason, you might have to first determine the ratios of natural isotopes *in the sample at hand* and, potentially, even for each compound class you are interested in.

7.7 Simulating isotopologues

Claude helped!

On an instrument with ultrahigh resolution, isotopologues with identical nominal mass may be resolved, see Sec. 7.5; the resulting pattern of peaks is also called the *isotopic fine structure*. To simulate it, we need the individual isotopologues instead of isotope peaks. Their number grows quickly, see (7.1), but almost all of them have negligible probability; so, we ask for all isotopologues with probability at least some threshold ε. A simple algorithm does the job: For each element E, we list all isotopologues of E_{i_E} with their probabilities (multinomial distribution) and masses, sorted by decreasing probability. An isotopologue of the molecule is a choice of one entry from each list; its probability is the product of the chosen probabilities, its mass the sum of the chosen masses. We choose entries recursively, one element after the other. Let p be the product of probabilities chosen so far, and let $p_{\max}$ be the product of the first (largest) probabilities of all lists not yet considered. If $p \cdot p_{\max} < \varepsilon$, then no extension can reach ε; since the list is sorted, the same holds for all remaining entries of the current list, and we stop the loop. Otherwise, choosing the first entry of all remaining lists results in an isotopologue with probability at least ε. Hence, every recursive call that passes the test produces at least one output, and the running time is $O(kL)$ for k elements and L reported isotopologues, plus the time for building and sorting the lists. For sucrose and $\varepsilon = 10^{-6}$, we find 27 of the 23 322 isotopologues, which together account for 99.9994 % of the probability mass. Things get algorithmically more complicated if we want to guarantee that the *sum of probabilities* of all isotopologues not in our output, is below a given threshold $\delta > 0$; yet, this is usually not required in practice. To simulate the fine structure, we finally merge isotopologues whose masses cannot be resolved by the instrument. As mentioned in Sec. 7.5, this in itself is a somewhat non-trivial endeavor.

7.8 Formal proof of the folding theorem

For the sake of completeness, we now provide a formal proof of Theorem 2. In fact, this proof is very simple. Readers not interested in the formal details can safely skip this section.

Let $\vec{N} = (N_1, \ldots, N_l) \in \mathbb{N}^l$ and $\vec{N}' = (N'_1, \ldots, N'_L) \in \mathbb{N}^L$ be vectors of nominal masses. We denote $\sum \vec{N} := \sum_{i=1}^{l} N_i$ and $\sum \vec{N}' := \sum_{i=1}^{L} N'_i$. Let $\vec{Y} := (Y_1, \ldots, Y_l)$ and $\vec{Y}' := (Y'_1, \ldots, Y'_L)$ be vectors of the input random variables, and note that

$$\mathbb{P}(\vec{Y} = \vec{N}, \vec{Y}' = \vec{N}') = \mathbb{P}(\vec{Y} = \vec{N})\mathbb{P}(\vec{Y}' = \vec{N}')$$

due to the independence of the underlying random variables. Finally, we set $\lambda(\vec{N}) = \sum_{i=1}^{l} \lambda_{E_i}(N_i)$ and analogously define $\lambda(\vec{N}')$.

We can rewrite (7.3) for the mass of the $+n$ peak as

$$\mathbb{P}(Z = N + N' + n) \cdot M_n = \sum_{\sum \vec{N} + \sum \vec{N}' = N+N'+n} \mathbb{P}(\vec{Y} = \vec{N}, \vec{Y}' = \vec{N}') \cdot \big(\lambda(\vec{N}) + \lambda(\vec{N}')\big).$$

We observe that we can split this formula into two independent sums of the form

$$\sum_{\sum \vec{N} + \sum \vec{N}' = N+N'+n} \mathbb{P}(\vec{Y} = \vec{N}, \vec{Y}' = \vec{N}') \cdot \lambda(\vec{N}) \tag{7.10}$$

and a second summand where $\lambda(\vec{N})$ is replaced by $\lambda(\vec{N}')$; we concentrate on (7.10) in the following. Now,

$$\begin{aligned}
&\sum_{\sum \vec{N} + \sum \vec{N}' = N+N'+n} \mathbb{P}(\vec{Y} = \vec{N}, \vec{Y}' = \vec{N}') \cdot \lambda(\vec{N}) \\
&\quad = \sum_{j=0}^{n} \sum_{\sum \vec{N} = N+j} \sum_{\sum \vec{N}' = N'+n-j} \mathbb{P}(\vec{Y} = \vec{N}) \mathbb{P}(\vec{Y}' = \vec{N}') \cdot \lambda(\vec{N}) \\
&\quad = \sum_{j=0}^{n} \sum_{\sum \vec{N} = N+j} \mathbb{P}(\vec{Y} = \vec{N}) \cdot \lambda(\vec{N}) \sum_{\sum \vec{N}' = N'+n-j} \mathbb{P}(\vec{Y}' = \vec{N}') \\
&\quad = \sum_{j=0}^{n} \sum_{\sum \vec{N} = N+j} \mathbb{P}(\vec{Y} = \vec{N}) \cdot \lambda(\vec{N}) \cdot \mathbb{P}(Y_1' + \cdots + Y_L' = N' + n - j) \\
&\quad = \sum_{j=0}^{n} \mathbb{P}(Y' = N' + n - j) \sum_{\sum \vec{N} = N+j} \mathbb{P}(\vec{Y} = \vec{N}) \cdot \lambda(\vec{N}) \\
&\quad = \sum_{j=0}^{n} q_{n-j} p_j m_j
\end{aligned}$$

where the last equality follows from the definition of m_j,

$$m_j = \frac{1}{p_j} \sum_{\sum \vec{N} = N+j} \mathbb{P}(\vec{Y} = \vec{N}) \cdot \lambda(\vec{N}).$$

Analogously, we can show that

$$\sum_{\sum \vec{N} + \sum \vec{N}' = N+N'+n} \mathbb{P}(\vec{Y} = \vec{N}, \vec{Y}' = \vec{N}') \cdot \lambda(\vec{N}') = \sum_{j=0}^{n} q_{n-j} p_j m'_{n-j}.$$

This concludes the proof of the theorem. □

7.9 Historical notes and further reading

I sincerely hope that nobody feels hurt by the name of the smart convolution algorithm. Calling someone "smart" should not be an issue, but political correctness can sprout some strange flowers.[7]

[7] I was once accused of being rude and insensitive because I used the German term "Milchmädchenrechnung" in a talk, which roughly translates to "back-of-the-envelope calculation" and literally means "milkmaid calculation". (According to Wikipedia, the German term is either from the fable "La Laitiére et le Pot au Lait" by Jean de La Fontaine, or the historic person Anna Schnasing, who happened to become a Direktrice in accounting for the dairy C. Bolle in Berlin around 1885.) Not only is the German term much more beautiful, it also means something slightly different, see the fable for details.

The formalism and methods presented in this chapter follow the paper of Böcker, Letzel, Lipták and Pervukhin [31], but note that some variable names have been changed: Here, we use n, N for the nominal masses of the molecule, whereas k is the size of the alphabet.

Early computational approaches for simulating isotope distributions were presented by Yergey [372] and Hsu [155]. In 1991, Kubinyi [195] suggested computing isotope distributions by convolving isotope distributions of "hyperatoms" using, in principle, the smart Russian algorithm from Algorithm 7.1.

In the literature on simulating isotope distributions and patterns, one can find many contributions by Alan L. Rockwood, for example using the Fourier transform to speed up computations [293, 295, 296]. Rockwood *et al.* [297] used mean peak masses as the masses of isotope peaks, presented some validation of this approach, as well as an algorithm for computing mean peak masses, which is more complicated and less efficient than the algorithm from Sec. 7.3. In 2006, Rockwood and Haimi [292] and Böcker, Letzel, Lipták and Pervukhin [30] independently came up with the algorithm presented in Sec. 7.3.

Computing the isotopic fine structure has received considerable attention. Snider [331] uses dynamic programming and keeps only the most probable exact masses; Li *et al.* [213] represent the isotopic species of a molecule in a hierarchical structure of transitions; Ipsen [161] uses the multidimensional Fourier transform; and Loos *et al.* [216] derive the relevant isotopologues of each element using pruned transition trees, and then combine them. IsoSpec [197, 198] can stop as soon as the reported isotopologues reach a given total probability, instead of using a threshold for individual isotopologues. Kreitzberg *et al.* [193] compute a given number of most abundant isotopologues using selection on Cartesian products. Dittwald *et al.* [82] study basic characteristics of the fine isotopic distribution. Be aware that all of this is mostly of theoretical interest: For small molecules, the number of relevant isotopologues is small, and simple algorithms such as the one from Sec. 7.7 are fast enough. For proteins, everything said in Sec. 7.6 about isotope patterns applies to isotopologues even more. Claude helped!

We have seen in Chapter 2 that we often record the fragmentation pattern of a molecule, to obtain additional information about its structure. Selecting only the monoisotopic peak for fragmentation, as is usually done for metabolites and other small molecules, substantially simplifies the interpretation of the fragmentation spectra: All fragments are purely monoisotopic, too. But what if we select, say, the monoisotopic and the +1 peak for simultaneous fragmentation? Obviously, the isotope distribution of fragments is *not* the isotope distribution of a "regular" molecule. Somewhat surprisingly, it is not too complicated to simulate these isotope distributions, see Rockwood, Kushnir and Nelson [294] and Exercise 7.13. On the downside, simulating such truncated isotope distributions requires considerably more time than the algorithms from Sec. 7.3. Also, it requires detailed information about the *mass filter* used as the first step of the MS/MS measurement. In contrast, if one opens the precursor mass window wide enough so that all "important" isotope peaks are selected for fragmentation, then isotope distributions of fragments *will* follow the isotope distribution as defined in Sec. 7.2. But this will increase the chance that besides the isotope peak of the molecule of interest, other molecules may "sneak" into the fragmentation process.

Masses of isotopes are taken from the atomic mass evaluation AME2020 [360] and rounded to six decimal places. Do not expect a new atomic mass evaluation to change your results, though: From AME2016 [359] to AME2020, the masses of all isotopes in Table 7.1 changed by at most $7 \cdot 10^{-9}$ Da (for ^{36}S) and by at most 0.34 ppb (for ^{1}H); rounded to six decimal places, they are identical. The mass of sucrose "changed" by about 10^{-8} Da. I am not aware of any mass spectrometer that would notice. Isotope abundances and atomic weights (average masses) are taken from the IUPAC technical reports of 2013 [238, 239]. See de Laeter *et al.* [78] for the history of atomic-weight determination. Claude helped!

The masses given in this chapter are not meant for use in computer programs, but rather for the human reader. This will only become an issue if you want to analyze spectra with mass accuracy below 1 ppm; but it is nevertheless a good idea to make your computations "as accurate as possible". You can download masses with higher mass accuracy from the Internet.[8] Computations in this chapter use masses from Table 7.1; in contrast, Table 2.1 has been computed using masses with higher accuracy. For example, an aspartic acid residue has mass 115.026943 according to Table 2.1, whereas $C_4H_5NO_3$ has mass

[8] https://www-nds.iaea.org/amdc/, https://www-nds.iaea.org/amdc/ame2020/mass_1.mas20.txt

```
1:  function DISTRIBUTION(array P of isotope distributions, integer l)
2:      isotope distributions Q = Q[0 ... n_max − 1] and Q′ = Q′[0 ... n_max − 1]
3:      if l ≤ L then
4:          return isotope distribution P[l]
5:      end if
6:      i ← ⌊l/L⌋; l′ ← l − iL
7:      Q ← P[l′]
8:      Q′ ← P[L]
9:      while i > 0 do
10:         if i is odd then
11:             Convolve Q and Q′, store result in Q
12:         end if
13:         Convolve Q′ and Q′, store result in Q′
14:         i ← ⌊i/2⌋
15:     end while
16:     return isotope distribution Q
17: end function
```

Algorithm 7.3: What to do when too many guests arrive: Computing the isotope distribution of E_l for $l > L$. The two-dimensional array P has been computed during preprocessing. Here, $P[l]$ is the isotope distribution for molecular formula E_l, for $l = 0, \ldots, L$, where $P[0] = (1, 0, \ldots, 0)$. Each distribution $P[l]$ consists of $n_{\max}$ entries $P[l, 0], \ldots, P[l, n_{\max} - 1]$. Convolve isotope distributions using (7.8).

115.026944 according to Table 7.1. Outside of this chapter, all masses are also computed with higher accuracy, except where we explicitly use the masses from Table 7.1, such as for the blowup factors in Sec. 8.2.

See Table 7.5 for isotope masses of elements less abundant in biomolecules: The halogens fluorine and chlorine are relatively common in small molecules, whereas iodine and bromine are less common. Selenium is part of the (uncommon but proteinogenic) amino acid selenocysteine; 25 human proteins are known that contain selenium [301]. Some metals such as copper or iron do not form covalent bonds in small molecules and, hence, will not be measured together with the molecule [236]. But for heme and other coordination complexes, the metal atom or ion can be measured together with the compound; I have included iron in Table 7.5 for heme and hemoproteins. Mercury and tin form covalent bonds in certain organometallic compounds, such as antifouling agents or fungicides [236]; I did not include them in the table, as you will not find many such compounds in any structure database. Sodium (Na) is included as it can form adduct ions such as $[M+Na]^+$ with the molecule M; the same holds true for potassium (K). Tungsten (W) is only present in this table because it is the heaviest element known to be biologically functional (essential for some archaea) and since its isotope pattern is rather special.

7.10 Exercises

7.1 Consider the isotopologues $^{13}C_{345}{}^{1}H_{344}$ and $^{12}C_{345}{}^{2}H_{344}$ for molecular formula $C_{345}H_{344}$. What is the probability of these two isotopologues? How many isotopologues will be present if you have 1 kg of pure substance?

7.2 Imagine a “sulfur-only” molecule — how large does this molecule have to be, so that the +10 peak has an intensity of more than 1 %? This can be seen as a worst-case scenario. For your computation, assume that sulfur has only two isotopes, namely nominal mass 34 with relative abundance $1 - p = 0.0425$, and nominal mass 32 with relative abundance p. Estimate the required number of sulfur atoms using (7.4). Be reminded that the heavier isotope of sulfur has nominal mass 34, not 33.

element (symbol)	AN	isotope	abundance %	mass (Da)	av. mass (Da)
boron (B)	5	^{10}B	19.9* %	10.012937	
		^{11}B	80.1* %	11.009305	10.81
fluorine (F)	9	^{19}F	100 %	18.998403	18.998403
sodium (Na)	11	^{23}Na	100 %	22.989769	22.989769
silicon (Si)	14	^{28}Si	92.223 %	27.976927	
		^{29}Si	4.685 %	28.976495	
		^{30}Si	3.092 %	29.973770	28.085
chlorine (Cl)	17	^{35}Cl	75.76 %	34.968853	
		^{37}Cl	24.24 %	36.965903	35.45
potassium (K)	19	^{39}K	93.258 %	38.963706	
		^{41}K	6.730 %	40.961825	39.0983
iron (Fe)	26	^{54}Fe	5.845 %	53.939608	
		^{56}Fe	91.754 %	55.934936	
		^{57}Fe	2.119 %	56.935392	
		^{58}Fe	0.282 %	57.933274	55.845
selenium (Se)	34	^{74}Se	0.86 %	73.922476	
		^{76}Se	9.23 %	75.919214	
		^{77}Se	7.60 %	76.919914	
		^{78}Se	23.69 %	77.917309	
		^{80}Se	49.80 %	79.916522	
		^{82}Se	8.82 %	81.916700	78.971
bromine (Br)	35	^{79}Br	50.69 %	78.918338	
		^{81}Br	49.31 %	80.916288	79.904
iodine (I)	53	^{127}I	100 %	126.904473	126.904473
tungsten (W)	74	^{180}W	0.12 %	179.946713	
		^{182}W	26.50 %	181.948206	
		^{183}W	14.31 %	182.950224	
		^{184}W	30.64 %	183.950933	
		^{186}W	28.43 %	185.954365	183.84

Table 7.5: Natural isotope abundances of elements less frequent in biomolecules. 'AN' is atomic number. Masses rounded to six decimal places. *Distribution of boron shows a strong variation, depending on where the sample is taken. Remember not to use these numbers in your programs.

7.3 Write a program to simulate the isotope distribution of an arbitrary molecular formula over the elements CHNOPS. Compute the isotope distribution of sucrose, and verify your result using Table 7.3.

7.4 Verify your calculations from Exercise 7.2 using the program from Exercise 7.3.

7.5 We noted above that among all entries in the KEGG COMPOUND database (release 42.0) with elements CHNOPS and mass below 3000 Da, not a single molecule has an intensity of the +10 peak larger than 0.007 %. But that version of the database is totally outdated by now — possibly, there are new molecular formulas in the current release that have +10 peaks of higher intensities?

7.6 In Algorithm 7.1 (the smart Russian convolution) you can get rid of two convolutions — how?

7.7 How many atoms of any element from CHNOPS are needed so that the +1 peak is more intense than the monoisotopic peak? For example, for C_{93} the relative intensity of the monoisotopic peak is 36.77 %, and 36.99 % for the +1 peak. How many atoms are needed so that the monoisotopic peak has less than 5 % intensity of the +1 peak?

7.8 In the mass spectrometry instrument, we do record only a finite number of molecules resulting in the observed isotope distribution; whereas in our calculations, we basically assumed an infinite number. Let us put some rigor in here: Assume that we have measured exactly N molecules, and we want to know what measures we can expect for the $+k$ peak. Show how to estimate the median, quantiles, standard deviation or any other statistics you can think of for the observed intensity of the $+k$ peak. Hint: They can be found in Wikipedia and any statistics textbook, but you first have to simulate the "ideal" isotope distribution.

7.9 A peptide $s \in \Sigma^*$ is said to be *pure* if it is made from repetitions of a single amino acid, $s = x^l$ for some $x \in \Sigma$. Find the pure peptide with intensity of the +10 peak larger than 1 % such that $\mu(s)$ is minimum. Also find the pure peptide of minimum mass whose monoisotopic peak has relative intensity below 5 %.

7.10 Let $s \in \Sigma^*$ be any peptide with intensity of the +10 peak larger than 1 % such that $\mu(s)$ is minimum. This peptide is not necessarily pure. Argue why its mass will be close to the mass calculated in the previous exercise.

7.11 Assume an isotopic labeling experiment where part of the carbon in each molecule has been replaced by "all ^{13}C". Let $h_1, \ldots, h_k$ be the isotope intensities measured by the instrument of the query molecule with *known* molecular formula over l elements. Show how to estimate the enrichment ratio in $O(k)$ time if your data was ideal. (Hint: This is a very simple computation.) Why should you not use this method in practice? The number may nevertheless serve as a start for a subsequent, more robust estimation procedure.

★ 7.12 Given two sorted lists corresponding to sets $A, B \subseteq \mathbb{R}$. Give an algorithm that computes the sorted list for set $C := \{a + b : a \in A, b \in B\}$ in time $O(|A| \cdot |B|)$. You may assume that all $a + b$ are different (in which case $|C| = |A| \cdot |B|$), or take into account in your algorithm that some elements might be identical. (Warning: This is an open problem in theoretical computer science.)

★ 7.13 Assume that not only the monoisotopic peak is picked for fragmentation, but also +1 and +2 peaks. Now, fragments will show a truncated isotope pattern, which is obviously not the full isotope pattern. Let f_p, f be the molecular formulas of the precursor molecule and the fragment, and choose f' such that $f + f' = f_p$. (Here, f' is the neutral loss, see Chapter 9.) Let Y, Y', Z be the random variables for f, f', f_p, and let N, N', N_p be the corresponding nominal masses. Assume we have picked peaks $0, \ldots, n_{\max} - 1$ from the precursor isotope distribution, or a subset thereof. Then, we can limit our calculations to the first $n_{\max}$ peaks of the truncated fragment distributions — explain why. We define a matrix $C[0 \ldots n_{\max} - 1, 0 \ldots n_{\max} - 1]$ by

$$C[i, j] := \mathbb{P}(Y = N + i) \cdot \mathbb{P}(Y' = N' + j).$$

Then, $\mathbb{P}(Z = N_p + n) = \sum_{j=0}^{n} C[j, n-j]$ holds. Describe an algorithm that computes the truncated isotope distribution of fragment f using matrix C and the above equation.

★ 7.14 Try to estimate the abundances of CHNOPS from some database for peptide mass spectra.

7.15 Describe an algorithm that convolves two lists of isotopologues.

8 Decomposing Isotope Patterns

"When you hear hoofbeats, think of horses not zebras." (Theodore Woodward)

Assume that we have measured an isotope pattern, and we want to find those molecular formulas that show the highest similarity to the measured pattern, over some fixed alphabet of elements. Note that the formal definition of "isotope pattern" depends, to a certain extent, on the application and the used MS technique, see Sec. 7.2 and 7.5. Unfortunately, decomposing an isotope pattern is a somewhat ill-posed problem, and we are not aware of any practical approaches that directly address this problem. Instead, we circumvent the problem, similar to the two-step strategy proposed in Sec. 2.8: First, we filter the set of molecular formulas to a manageable subset, using only one or a few features of isotope patterns, in particular the monoisotopic mass. This leaves us with a set of candidate molecular formulas. The first step is not meant to differentiate between the candidates; its only purpose is to quickly generate a candidate set of manageable size. In the next step, we evaluate the candidates using the isotope patterns: As we now have a candidate molecular formula, it is an easy task to simulate the corresponding isotope pattern using methods from Sec. 7.3, and to compare the simulated isotope pattern against the measured one, comparable to a database search. The candidate with the best match against the measured isotope pattern is the output of our method, and hopefully the correct answer.

Our input is a list of masses $M_0, \dots, M_{n_{\max}}$ with intensities $h_0, \dots, h_{n_{\max}}$. We assume that these have been extracted from a mass spectrum in a preprocessing step, and that they correspond to the isotope pattern of a single sample molecule. Separating isotope peaks that belong to different molecules is trivial for most cases. Our goal is to find the molecular formula whose isotope pattern best matches the input. Given a molecular formula candidate, we can predict its spectrum with masses $m_0, \dots, m_{n_{\max}}$ and intensities $g_0, \dots, g_{n_{\max}}$ as described in the previous chapter.

Even though MS instruments record ions, we will sometimes consider neutral molecules in our presentation. This simplifies matters, but does not restrict the method in any way.

What about peptides, that is, decomposing over the alphabet of amino acids? As we will see in Sec. 8.6, it is not a clever idea to decompose over this alphabet directly. The amino acid alphabet is simply too large, leading to a huge number of decompositions with identical molecular formula and, hence, identical simulated isotope patterns. In contrast, one can directly decompose over small alphabets: For glycans, we can often assume a small alphabet with only three or four simple sugars, see Chapter 11.

8.1 Decomposing real numbers

We now come back to the problem of decomposing a real number, namely, the monoisotopic mass M_0. In Chapter 3 we have seen how to efficiently decompose integers, and we want to utilize these methods to do the same thing for real numbers. When decomposing real numbers, we have to take into account the inaccuracy of MS measurements: We want to find all molecular formulas with monoisotopic mass in the interval $[l', u'] \subseteq \mathbb{R}$ where $l' := M_0 - \varepsilon$ (lower bound) and $u' := M_0 + \varepsilon$ (upper bound) for some measurement inaccuracy ε. Formally, we search for all solutions of the equation

$$\mu'(c) := a'_1 c_1 + a'_2 c_2 + \cdots + a'_k c_k \in [l', u'] \,, \tag{8.1}$$

where $a'_1, \dots, a'_k$ are the real-valued monoisotopic masses of elements. We search for all compomers $c = (c_1, \dots, c_n)$ satisfying (8.1) or, equivalently, $\mu'(c) \in [l', u']$. Searching for compomers c with $\mu'(c) = M_0$ does not make sense in the real-valued setting: This set is practically always empty. Again, we may assume $a'_1 < a'_2 < \cdots < a'_n$.

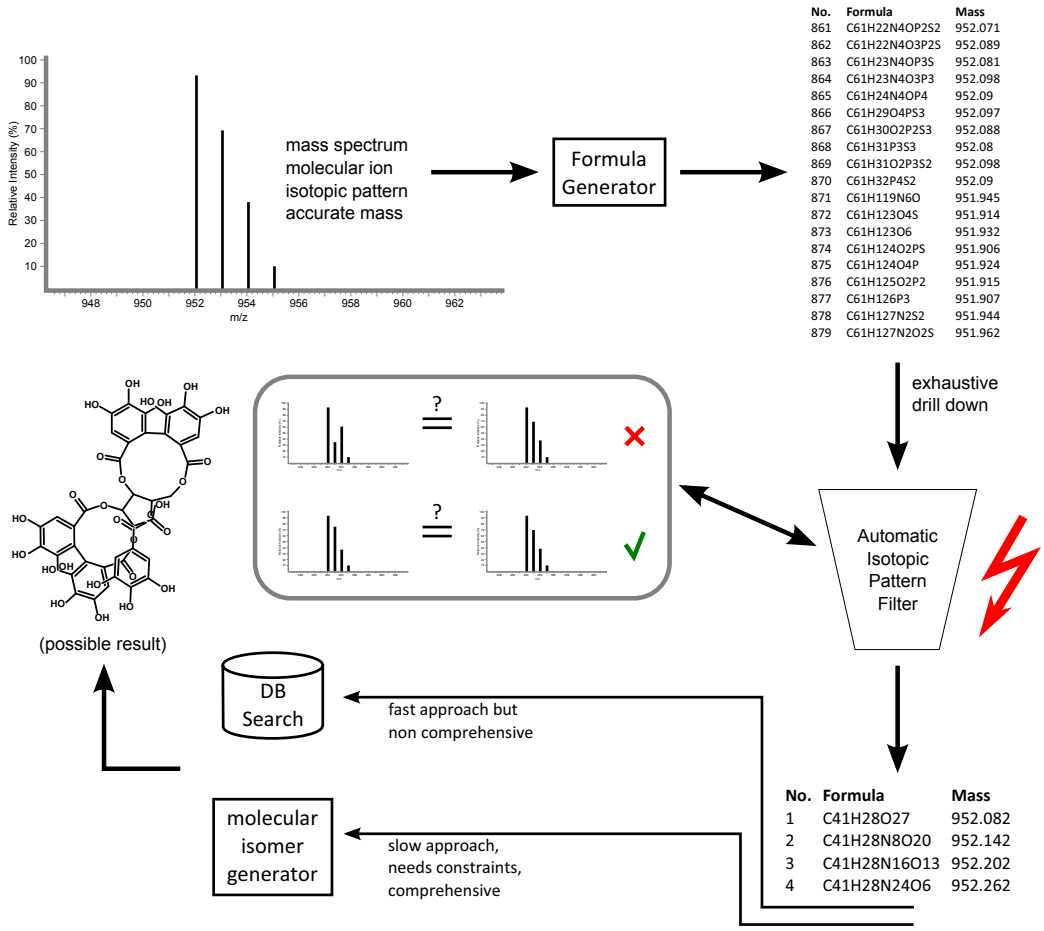


Figure 8.1: Decomposing isotope patterns using a two-step approach: First, molecular formulas are filtered using the monoisotopic mass of the molecule. Second, candidate molecular formulas are filtered using the full isotope pattern. Figure taken from [309].

A straightforward solution to this problem, is to enumerate all vectors c with $c_1 = 0$ and $\sum_j a'_j c_j \leq u'$, and next to test if there is some $c_1 \geq 0$ such that $\sum_j a'_j c_j \in [l', u']$. For readability, we will omit the limits of the sum in case these limits are obvious: Here, $j = 1, \ldots, k$. We can do so by nesting $|\Sigma| - 1$ For-loops. An algorithm that works for an alphabet of arbitrary size, and avoids the nasty nesting, is given in Algorithm 8.1. This results in $\Theta(M_0^{k-1})$ running time, which is acceptable in applications if you want to decompose only a few numbers. But often, you want to use the decomposition algorithm as a subroutine of a larger algorithm, see for example Chapters 9 and 11. Then, this subroutine might be executed thousands of times. Here, improving the running time from one second to one millisecond, will have a large impact on the overall running time of the algorithm.

Another approach is to compute all potential decompositions up to some upper bound U during preprocessing, sort them with respect to mass and use binary search; this results in $\Theta(U^k)$ space requirement, but only requires $k \log U$ time for searching, using binary search. Both presented approaches are unfavorable in theoretical complexity as well as in practice: For the elements CHNOPS there exist more than $7 \cdot 10^9$ molecular formulas with mass below 1500 Da. It is somewhat strange to dedicate many gigabytes of memory to a subroutine, when the same problem can be solved with about one megabyte.

The above estimates are for the CHNOPS alphabet of elements: We are facing a combinatorial explosion if we add only two or three more elements. Table 7.5 on page 126 lists several "uncommon" elements that, depending on the application you have in mind, have to be added to the alphabet: For example, you should consider halogens such as chlorine or bromine as part of your alphabet if you are analyzing pharmaceutical small molecules.

```
1:  procedure FindAllNaive(real-valued lower bound l′, upper bound u′)
2:      compomer c ← 0
3:      mass M ← 0
4:      integer i ← 1                                   ▷ to get the following loop started
5:      while i ≤ k do
6:          for c_1 ← ⌈(l′−M)/a′_1⌉, ..., ⌊(u′−M)/a′_1⌋ do      ▷ this loop may be empty
7:              Output c
8:          end for
9:          M ← M + a′_2; c_2 ← c_2 + 1; i ← 2
10:         while M > u′ and i ≤ k do
11:             M ← M − c_i a′_i; c_i ← 0                ▷ clear less significant "digits"
12:             i ← i + 1                                ▷ increase next "digit"
13:             if i ≤ k then
14:                 M ← M + a′_i; c_i ← c_i + 1
15:             end if
16:         end while
17:     end while
18: end procedure
```

Algorithm 8.1: Naïve but, at least, iterative algorithm for enumerating all compomers c with $\mu'(c) \in [l', u']$. Constants $a'_1, \ldots, a'_k$ are the real-valued masses.

In the remainder of this section, we transform the enumeration problem with real-valued masses into a problem instance with *integer* masses. We already noted that we can transform real-valued masses to integer masses by multiplying all masses by a large constant, and rounding the results. We now formalize this idea: Given a *blowup factor* $b \in \mathbb{R}$, we round coefficients by $\lfloor bx \rfloor$, where $\lfloor \cdot \rfloor$ denotes the floor function for rounding down. Blowup factor b corresponds to precision $1/b$ of our calculations. This precision $1/b$ is merely a parameter of the decomposition algorithm, and independent of the measurement mass accuracy ε. We will discuss selecting a "good" blowup factor in the next section.

We transform all real-valued masses $a'_1, \ldots, a'_k$ into integer masses $a_i := \lfloor ba'_i \rfloor$, and we also calculate integer bounds $l := \lfloor bl' \rfloor$ and $u := \lfloor bu' \rfloor$. We want to find all compomers c with $\mu(c) \in [l, u]$ over the integer alphabet $\Sigma = \{a_1, \ldots, a_k\}$, where $\mu(c) := \sum_j c_j a_j$ denotes the weighting function for integer weights. This can be achieved by iterating over $M = l, \ldots, u$, and enumerating all c with $\mu(c) = M$ for each M. In Sec. 3.5 and 3.6, we have presented methods for efficiently doing so.

Does this already solve our problem? Unfortunately, no: Certain solutions c of the integer mass instance are not solutions of the real-valued mass instance, and vice versa. In other words, there might be compomers c with integer mass $\mu(c) \in [l, u]$ but real-valued mass $\mu'(c) \notin [l', u']$. These are *false positive* solutions, as we would wrongly report them when solving the integer instance. We can easily sort out false positive solutions by checking (8.1) for every decomposition c, resulting in additional running time for computing some integer decompositions "in vain". On the other hand, there might be compomers c with integer mass $\mu(c) \notin [l, u]$ but real-valued mass $\mu'(c) \in [l', u']$. These are *false negative* solutions as we would wrongly omit them when solving the integer instance. We now concentrate on the more intriguing problem of false negative solutions.

For the upper bound, we want to show that $\mu'(c) \le u'$ implies $\mu(c) \le u$: This follows because

$$\sum_j c_j a_j = \sum_j \lfloor c_j a_j \rfloor \le \sum_j c_j b a'_j = b \sum_j c_j a'_j \le bu'$$

```
1:  procedure FindAll(real-valued lower bound l′, upper bound u′)
2:      a_j := ⌊ba′_j⌉ for j = 1, ..., k
3:      l := ⌈bl′ − Δl′⌉                          ▷ decreased bound, to avoid false negatives
4:      u := ⌊bu′⌋
5:      for M = l, ..., u do
6:          Enumerate all compomers c with μ(c) = M over the alphabet a_1, ..., a_k
7:          for each compomer c with μ(c) = M do
8:              if μ′(c) ≥ l′ and μ′(c) ≤ u′ then          ▷ remove false positives
9:                  Output c
10:             end if
11:         end for
12:     end for
13: end procedure
```

Algorithm 8.2: Smart algorithm for enumerating all compomers c with $\mu'(c) \in [l', u']$. Constants $a'_1, \ldots, a'_k$ are the real-valued masses. Blowup factor b is a constant, and Δ is computed from (8.2). The integer masses $a_1, \ldots, a_k$ are also constant, and line 2 is only meant to remind the reader of their definition.

and, as $\sum_j c_j a_j$ is integer, $\mu(c) = \sum_j c_j a_j \leq \lfloor bu' \rfloor = u$. This means that we do not have to change the upper bound u, as we will never miss a real-valued decomposition of (8.1) here. On the other hand, we have to decrease the integer lower bound l to guarantee the same thing: We define relative rounding errors

$$\Delta_j = \Delta_j(b) := \frac{ba'_j - \lfloor ba'_j \rceil}{a'_j} = b - \frac{\lfloor ba'_j \rceil}{a'_j} \quad \text{for } j = 1, \ldots, k, \tag{8.2}$$

and note that $0 \leq \Delta_j < \frac{1}{a'_j}$. Let $\Delta = \Delta(b) := \max_j\{\Delta_j\}$. We claim:

$$\sum\nolimits_j a'_j c_j \geq l' \quad \text{implies} \quad \sum\nolimits_j a_j c_j \geq bl' - \Delta l' \tag{8.3}$$

So, assume that $\sum_j a'_j c_j \geq l'$ holds. Our claim follows from

$$\begin{aligned}
\sum\nolimits_j a_j c_j &= \sum\nolimits_j ba'_j c_j - \sum\nolimits_j \left(ba'_j - a_j\right) c_j \\
&= b \sum\nolimits_j a'_j c_j - \sum\nolimits_j \left(ba'_j - \lfloor ba'_j \rceil\right) c_j \\
&= b \sum\nolimits_j a'_j c_j - \sum\nolimits_j \frac{ba'_j - \lfloor ba'_j \rceil}{a'_j} a'_j c_j \\
&= b \sum\nolimits_j a'_j c_j - \sum\nolimits_j \Delta_j a'_j c_j \\
&\geq b \sum\nolimits_j a'_j c_j - \Delta \sum\nolimits_j a'_j c_j \\
&= (b - \Delta) \sum\nolimits_j a'_j c_j \\
&\geq (b - \Delta)\, l' = bl' - \Delta l',
\end{aligned}$$

where the last step uses $b - \Delta > 0$, which holds as $\Delta < 1 \leq b$ for masses $a'_j \geq 1$ and blowup factor $b \geq 1$. This implies that we can use $l := \lceil bl' - \Delta l' \rceil$ as the lower bound of the integer decomposition instance.

Algorithm 8.2 shows how to decompose a real-valued mass. Line 8 of the algorithm ensures that we will never output any false positives, and Eq. (8.3) guarantees that this algorithm will never miss a decomposition.

Can we save some time by slightly increasing the new lower bound $\lceil bl' - \Delta l' \rceil$? It turns out that this is not possible, as the new lower bound is *tight*: That is, no larger bound can be chosen without missing

some real-valued decompositions. Precisely speaking, given an arbitrary real-valued alphabet, we can find an infinite number of compomers c and lower bounds l' such that $\mu'(c) \geq l'$ (true decomposition for real-valued masses) but $\mu(c) = \lceil bl' - \Delta l' \rceil$. To show this, assume that $\Delta = \Delta_j$ for some element E with index j. Consider the decomposition $c = q \cdot e_j$ for some integer $q > 0$. Recall that $e_j = (0, \ldots, 0, 1, 0, \ldots, 0)$ denotes the jth unit vector that has all-zero entries, except for the jth entry, which equals one. If we set the lower bound of our decomposition as $l' := \mu'(c) = qa'_j$ then c is part of the output. To ensure this, $l \leq \mu(c) = qa_j$ must hold. We calculate

$$bl' - l \geq bqa'_j - qa_j = q \cdot \left(ba'_j - \left\lfloor ba'_j \right\rfloor \right) = qa'_j \frac{ba'_j - \left\lfloor ba'_j \right\rfloor}{a'_j} = qa'_j \Delta_j = l' \Delta.$$

Hence, $l \leq bl' - \Delta l'$ and, as l is integer, $l \leq \lfloor bl' - \Delta l' \rfloor$ as claimed.

As indicated, decreasing the lower bound forces us to decompose a larger number of integer masses: Without rounding correction we have to decompose about $(u' - l')b$ integer masses, but rounding correction forces us to decompose an additional $\Delta l'$ integer masses, independent of the interval size $u' - l'$. (For readability, we have ignored the effect of rounding when estimating these numbers, as this has a negligible impact.) This appears to be somewhat unfortunate: Even if ε is very small, the number of integers we have to decompose is linear in the mass M_0 of the measurement. But we should keep in mind that mass accuracy gets worse with increasing mass, and is measured as a relative value such as $\alpha = 10\,\text{ppm} = 10^{-5}$, see Sec. 4.7. So, the absolute accuracy is itself a linear function in M_0, $\varepsilon(M_0) := \alpha \cdot M_0$. The number of integer masses we have to decompose, then becomes roughly $(2\alpha b + \Delta)M_0$. Also, be reminded that the running time for decomposing integer masses (Algorithm 3.4) is dominated by the *number of decompositions* of these integers, and not by the number of integers itself.

Example 8.1. Consider the weighted alphabet of elements Σ = CHNOPS and blowup factor $b = 10^4$. Using masses from Table 7.1, we compute

$$\begin{array}{lll} \Delta_{\text{C}}(b) = 0 & \Delta_{\text{H}}(b) = 0.2481 & \Delta_{\text{N}}(b) = 0.0528 \\ \Delta_{\text{O}}(b) = 0.0094 & \Delta_{\text{P}}(b) = 0.0200 & \Delta_{\text{S}}(b) = 0.0222 \end{array}$$

where $\Delta_x(b)$ denotes the relative rounding error of character $x \in \Sigma$. So, $\Delta(b) = \Delta_{\text{H}}(b) = 0.25 \,/\, 1.007825 = 0.2481$. Assume that we want to decompose the real-valued mass $M_0 = 1000$. Then, we have to decompose an additional 248 integers, independent of the mass accuracy. For mass accuracy 10 ppm we have $\varepsilon = 0.01$ and $u' - l' = 0.02$. In total, we have to decompose 449 integer masses, instead of 201 without correction.

8.2 Good choices for the blowup factor

Algorithm 8.2 tells us how to decompose any interval of real numbers; the only parameter of this approach that we have not considered, is the blowup factor b. Be reminded that independent of the choice of b, Algorithm 8.2 will never miss a molecular formula, or produce a false positive. In practice, you would choose b "reasonably": It should not be too small, taking into account the anticipated mass accuracy of the instrument. Otherwise, many integer decompositions will be computed in vain, and have to be discarded by line 8 of the algorithm. On the other hand, it should not be too large: Even though computers have gigabytes of memory these days, accessing this memory is significantly slower than accessing the processor cache, just like accessing the hard disk is significantly slower than accessing the internal memory. In practice, a comparatively small b appears to be a good choice, so that the extended residue table of Algorithm 3.4 uses less than one megabyte of memory; recall that the size of this table is $O(k \left\lfloor ba'_1 \right\rfloor)$.

We will now look at "good choices" for parameter b: Such blowup factors will result in a small quotient $\Delta(b)/b$ of additional integers we have to decompose. Note that we write $\Delta(b)$ here, to stress that Δ actually depends on the chosen blowup. We have to decompose a total of $(2\alpha b + \Delta(b))M_0$ integer masses, and

$\Delta(b)M_0$ of these are decomposed because of our rounding technique. We want to minimize the relative number of integers that have to be decomposed in addition, being

$$\frac{\Delta(b)M_0}{\big(2\alpha b + \Delta(b)\big)M_0} = \frac{\Delta(b)}{2\alpha b + \Delta(b)}, \tag{8.4}$$

and this number is minimum if $\Delta(b)/b$ is minimum. You can easily see this if you try to maximize the (multiplicative) inverse of (8.4). See, for example, Example 8.1: Choosing $b = 10^4$ seems to be a bad idea as $\Delta_\mathrm{H} \gg \Delta_x$ for $x \in \{\mathrm{C}, \mathrm{N}, \mathrm{O}, \mathrm{P}, \mathrm{S}\}$.

Suppose that memory considerations imply a maximum blowup factor of $B \in \mathbb{R}$. We want to find $b \in (0, B]$ such that $\Delta(b)/b$ is minimized. We think of $\Delta_j : \mathbb{R} \to \mathbb{R}$ as a function of b,

$$\Delta_j(b) = b - \frac{1}{a'_j}\left\lfloor ba'_j \right\rfloor,$$

for all $j = 1, \ldots, k$. Each Δ_j is a piecewise linear function with discontinuities $\frac{1}{a'_j}, \frac{2}{a'_j}, \ldots, \frac{\lfloor a'_j B \rfloor}{a'_j}$. In every interval, this function has slope 1. Next, we set $\varphi_1 \equiv \Delta_1$ and for $j \geq 2$, we define φ_j as the maximum of φ_{j-1} and Δ_j. Each φ_j is a piecewise linear function; and $\Delta \equiv \varphi_k$ is a piecewise linear function with $O\big((a'_1 + \cdots + a'_k)B\big)$ discontinuities. For every piecewise linear part $I \subseteq \mathbb{R}$ of Δ the minimum of $\frac{\Delta(b)}{b}$ must be located at one of the terminal points. We sweep over the discontinuities from left to right, and for each discontinuity b, we calculate all $\Delta_j(b)$ and $\Delta(b)$. See Dührkop *et al.* [92] for details, and Table 2 there for all locally optimal blowup factors.

It turns out that 1182.7510330 and 5963.3376861 are very good blowup factors, for the alphabets of elements CHNOPS *and* CHNOPSClBrI, as these blowups are locally optimal in both cases; for larger blowups, 44770.6696249 (CHNOPS) and 44770.6721964 (CHNOPSClBrI) are good choices. In practice, blowup $b = 5963.3376861$ appears to give a good balance between saving memory — which speeds up computations as we can store all tables in the processor cache — and generating not too many decompositions "in vain" [92].

Example 8.2. Consider the weighted alphabet of elements Σ = CHNOPS and blowup factor $b = 5963.3376861$. Using masses from Table 7.1, we compute:

$$\begin{aligned} \Delta_\mathrm{C}(b) &= 0.0044 \quad \Delta_\mathrm{H}(b) = 0.0008 \quad \Delta_\mathrm{N}(b) = 0.0042 \\ \Delta_\mathrm{O}(b) &= 0.0050 \quad \Delta_\mathrm{P}(b) = 0.0001 \quad \Delta_\mathrm{S}(b) = 0.0080 \end{aligned}$$

So, $\Delta(b) = \Delta_\mathrm{S} = 0.0080$, see Example 8.1. For mass 1000 we have to decompose an additional 8 integer masses.

8.3 Scoring isotope patterns

Now that we have filtered down to a few (possibly, still tens of thousands of) molecular formulas, we have to rank them. We can think of this as searching in a database, see Chapter 4 and in particular Sec. 4.10. Certain aspects should be considered in our scoring, though:

- Unlike matching general spectra, isotope patterns are rather boring with respect to the presence or absence of certain peaks: Any isotope pattern is a string of peaks at about one Dalton distance. Matching peak pairs between query and reference spectrum is basically trivial.
- The much bigger problem is to identify an isotope pattern in, say, an LC-MS run. In particular, it is non-trivial to detect the low-intensity peaks in the isotope pattern.
- We have a clear idea of peak intensities. All peaks correspond, chemically speaking, to the same molecule; hence, there should be no intensity variation through ionization preferences.

- All peaks are very close to each other. We can locally correct for "mass shifts".
- We have some chemical understanding of which candidates are "somewhat unlikely" and should be penalized, see Sec. 8.4 below.

We mostly use the ideas of Sec. 4.10 to build a scoring for isotope patterns. We only slightly modify the scoring, as suggested in Sec. 4.11. Recall that the score presented there is based on computing the likelihood of the data. We will instead use Bayes' rule and rank our candidates by *posterior probability* $\mathbb{P}(\mathcal{M}_i|\mathcal{D})$,

$$\mathbb{P}(\mathcal{M}_i|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{M}_i) \cdot \mathbb{P}(\mathcal{D}|\mathcal{M}_i)}{\mathbb{P}(\mathcal{D})} \tag{8.5}$$

where $\mathcal{D}$ is the data (the measured spectrum), $\mathcal{M}_i$ are the models (the candidate molecular formulas). The *prior probability* $\mathbb{P}(\mathcal{M}_i)$ is covered in Sec. 8.4; for the moment, we may assume a flat prior. Eq. (8.5) allows us to compute the probability of each model (given the data), using the probability of the data, given each model. It turns out that we do not have to compute the probability of the data, as

$$\mathbb{P}(\mathcal{D}) = \sum_j \mathbb{P}(\mathcal{M}_j)\, \mathbb{P}(\mathcal{D}|\mathcal{M}_j),$$

so it is sufficient to normalize probabilities $\mathbb{P}(\mathcal{M}_i) \cdot \mathbb{P}(\mathcal{D}|\mathcal{M}_i)$ to one. We now iterate over all models, and concentrate on a particular model $\mathcal{M}$ as our candidate molecular formula.

As peak picking may not pick the complete isotope pattern, we score only those peaks that were detected in the query spectrum, and we do not penalize missing peaks. We also do not penalize additional peaks, as these must be contaminants and can safely be ignored.

Compared to Sec. 4.10, we make the following modification when scoring peak masses. Recall that the mass accuracy α of the instrument is given as a parameter, such as $\alpha = 10\,\text{ppm} = 10^{-5}$. We assume some standard deviation for peak masses such as $\sigma_{\text{mass}} := \frac{1}{3}\alpha M_0$, assuming that more than 99.7 % of measurements fall into the specified mass range. We may also take into account that small peaks usually have worse mass accuracy than high-intensity peaks, using an individual mass accuracy for each peak that depends on the intensity of the corresponding peak; we omit the details. For the mass of the monoisotopic peak, we calculate the probability that the mass difference is at least this large, as in Sec. 4.7:

$$\mathbb{P}(M_0|m_0) = \operatorname{erfc}\left(\frac{|M_0 - m_0|}{\sqrt{2}\,\sigma_{\text{mass}}}\right) = \frac{2}{\sqrt{2\pi}} \int_z^\infty e^{-t^2/2} dt \tag{8.6}$$

with $z := \frac{|M_0 - m_0|}{\sigma_{\text{mass}}}$. For the following peaks, we can eliminate the mass bias: We do not compare masses of the +1, +2, ... peaks directly but instead, we compare the difference to the monoisotopic peak, $M_j - M_0$ vs. $m_j - m_0$:

$$\mathbb{P}(M_j|m_j) = \operatorname{erfc}\left(\frac{\left|M_j - M_0 - m_j + m_0\right|}{\sqrt{2}\,\sigma'_{\text{mass}}}\right) \tag{8.7}$$

for $j = 1, \ldots, n_{\max}$. Recall from Exercise 4.9 that from a mathematical standpoint, we would have to assume an increased standard deviation of $\sigma'_{\text{mass}} = \sqrt{2}\,\sigma_{\text{mass}}$; but from an MS perspective, it makes more sense to use a substantially smaller standard deviation $\sigma'_{\text{mass}} < \sigma_{\text{mass}}$, as we have removed the bias part of the mass error. Stochastically speaking, our assumption of independence between peak masses has become more realistic through removing the mass deviation bias.

Compared to Secs. 4.8 and 4.10, we make the following modification when scoring peak intensities. First, we truncate the isotope pattern of the candidate to the same length as the query isotope pattern. The other peaks of the query isotope pattern might get lost due to a multitude of reasons which we cannot control. We have seen that isotope distributions tend to "deteriorate quickly", see for example Table 7.3 on page 111. Second, we normalize intensities in both the query and the reference mass spectrum: Absolute intensities correspond to the abundance of the molecule, its ionization potential, other molecules in

the sample etc., all of which we have no knowledge about. There exist three possibilities to normalize intensities: We can normalize such that (i) $\max_i h_i = 1$, (ii) $h_0 = 1$, or (iii) $\sum_i h_i = 1$; analogously for the theoretical spectrum g. We stress that g has to be normalized at this point although it corresponds to an (isotope) distribution, because we have truncated peaks at the end. Normalizing the sum agrees with our intuition that this is a distribution which, by definition, sums up to one. On the other hand, one wrongly measured peak intensity will result in all other peak intensities wobbling. In practice, it appears that normalizing the first peak works better [90]. Böcker *et al.* [31] suggested also incorporating the theoretical intensity of the "first missing peak", but practical issues with peak detection methods speak against doing so.

8.4 Integrating chemical knowledge

Before we start, a word of warning: Priors for integrating background knowledge should be handled with extreme care, see Chapter 12.

We will now take another look at the molecular formulas we are generating: These, after all, should correspond to some molecules. Be reminded that we generate *all* molecular formulas, and consider not only those present in some database. First, let us consider the molecule graph corresponding to the molecule's structure: This is an undirected graph where nodes are labeled with elements, and multiple edges may exist between two nodes to indicate the bond order of the covalent bonds. This graph should be connected, corresponding to a molecule where all atoms are connected by covalent bonds. The element by which a node is labeled, determines the degree of the node: This is called the *valence* of the element. In fact, the term "valence" is somewhat ambiguous and the truth is more complicated; but for us, it serves as a useful concept to filter out "impossible" molecular formulas. In organic compounds, carbon has valence 4, hydrogen valence 1, and oxygen valence 2. Unfortunately, many elements do not have a single valence: Nitrogen has valence 3 in organic compounds, but can also have valence 5; phosphorus has valence 3 or 5; and sulfur has valence 2, 4 or 6. But even carbon sometimes has a non-standard ("abnormal") valence, see Table 2 in [358].

For a graph to be connected, it must have at least $n-1$ edges to connect the n nodes; a connected graph with exactly $n-1$ edges and n nodes is a tree. Furthermore, each edge connects exactly two nodes. For molecule graphs, this inspired the definition of the *Ring Double Bond Equivalent* (RDBE) value, also known as degree of unsaturation, see for example Dayringer and McLafferty [75]:

$$\text{RDBE} = 1 + \frac{1}{2}\sum_i c_i(val_i - 2)$$

where val_i is the minimum valence (lowest valence state) of element $\#i$. This number was meant to count the number of rings and double bonds in a molecule; unfortunately, this is not true. As some atoms can form more covalent bonds than indicated by the minimum valence, some molecules will have a negative RDBE: For example, methylenebis(pentafluorosulfur) with molecular formula $CH_2F_{10}S_2$ has RDBE -4, as it comprises sulfur at a valence state of six [190]. Fractional (non-integer) RDBE values correspond to radicals, which are chemically unstable and, hence, unlikely to be observed in our sample. For this reason, it has been suggested repeatedly in the literature to discard all molecular formula candidates which have fractional or negative RDBE. This is often referred to as "Senior's rules" or "Senior's theorem" [319]. Note that the RDBE changes through ionization; in particular, a singly charged protonated ion has RDBE $x - \frac{1}{2}$ where x is the RDBE of the corresponding molecule.

But should we indeed discard molecular formula candidates, as suggested above? I would argue that this is not necessary. Radicals will be observed very rarely in experiments, except for electron ionization where the molecular ion is a radical cation (p. 9).[1] But their mass, isotope pattern and fragmentation spectrum (see Chapter 9) will be very different from that of any non-radical molecule. Consequently,

[1] There is the additional complication of intrinsically charged molecules which can be observed in mass spectrometry without having to ionize them; for such molecules, the RDBE will be integer, incorrectly suggesting an ionized radical. Notably, I am

candidate molecular formulas of radicals will rarely (if ever) be ranked high if we have experimental data from a non-radical molecule. Instead of discarding the candidate, we may rather penalize a non-integer RDBE in our candidate scoring. Similarly, we should only discard candidate molecular formulas with negative RDBE* when the RDBE* has been calculated using the *maximum* valence of all elements; again, we can penalize candidates if the minimum valence-based RDBE is negative. Biomolecules ("molecules of biological interest") with negative RDBE are rare, but they exist: For example, about 0.16 % of substances in the KEGG COMPOUND database [171] (release 42.0) violate this rule.

Are there further considerations on what molecular formulas are "chemically impossible"? Kind and Fiehn [190] presented seven "rules" to discard molecular formulas which are very different from what we expect to see in biomolecules; one of them (rule 3) considers the measured isotope pattern of the molecule, and another one is specific to GC-MS. These "rules" are based on statistical observations and the distribution of certain values in biomolecule databases: For example, how many nitrogen atoms does a biomolecule with mass below 500 Da have? What is the hydrogen-to-carbon ratio of biomolecules? But in every filtering step, we discard some molecular formulas which correspond to known biomolecules. So, it again is more appealing to penalize "strange" candidate molecular formulas, instead of discarding them.

A word of warning on bad priors: We cannot use the empirical distributions of, say, the hydrogen-to-carbon ratio directly, to compute a sensible prior probability. These empirical distributions only describe what is found in some molecule databases. There is no information on how frequently each molecular formula is found in an MS experiment; in particular, there is no information about the experiment that *you* are analyzing. It might be that certain hydrogen-to-carbon ratios are quite common in the database, whereas the corresponding molecules are extremely rare in experiments. It might also be that you are experimentally looking at a particular class of biomolecules that has hydrogen-to-carbon ratios quite different from what you find in databases. For this reason, you should never give a bonus for a "good-looking" molecular formula, and indeed restrict yourself to penalizing those that are outliers. See Chapter 12 for more details on this subject.

Böcker and Dührkop [28] introduced several elaborate priors to penalize outlier molecular formulas; but Dührkop [89] later showed that only two priors are needed to reach a good separation: One prior is based on RDBE, the other on the ratio of atoms which are not carbon, hydrogen or oxygen, to carbon atoms. Even better separation was reached with a linear support vector machine, trained with molecular formulas from biomolecules as positive training examples and random molecular formulas as negative training examples [89].

However you decide to penalize outlier molecular formulas, it makes sense to keep a "whitelist": Those are molecular formulas which you (and your priors) assume to be outliers, but corresponding molecules are frequently observed in experiments. For example, $C_{10}HF_{19}O_2$ does not look like a reasonable molecular formula for a biomolecule, neither to me nor to the machine learning approach mentioned above; but this is the molecular formula of perfluorodecanoic acid, which is a frequent contaminant in certain metabolomics experiments.

8.5 On the number of molecular formulas over CHNOPS

In Chapter 3, we have described how to approximate the number of decompositions over an integer alphabet [15]. We use this to approximate the number of molecular formulas over the elements CHNOPS: We multiply all masses by a large blowup factor b and round. (Since we are free to choose a blowup factor, we can make sure that the integer masses of elements have greatest common divisor one.) Assuming that

not talking about ions such as choline (which chemists call "permanently charged") with molecular formula $C_5H_{14}NO^+$, an essential nutrient for animals: Our mass decomposition would first subtract a proton, then find a decomposition $C_5H_{13}NO$. Understanding that molecular formula $C_5H_{14}NO^+$ is the one we are interested in, and that this ion is not a protonated form of some uncharged molecule $C_5H_{13}NO$ (although such molecules exist, for example 3-(dimethylamino)propan-1-ol), has to be done in the downstream analysis. Unfortunately, I am still unsure if intrinsically charged molecules (integer RDBE, odd charge) even exist. But I have come to believe that this is only a misunderstanding.

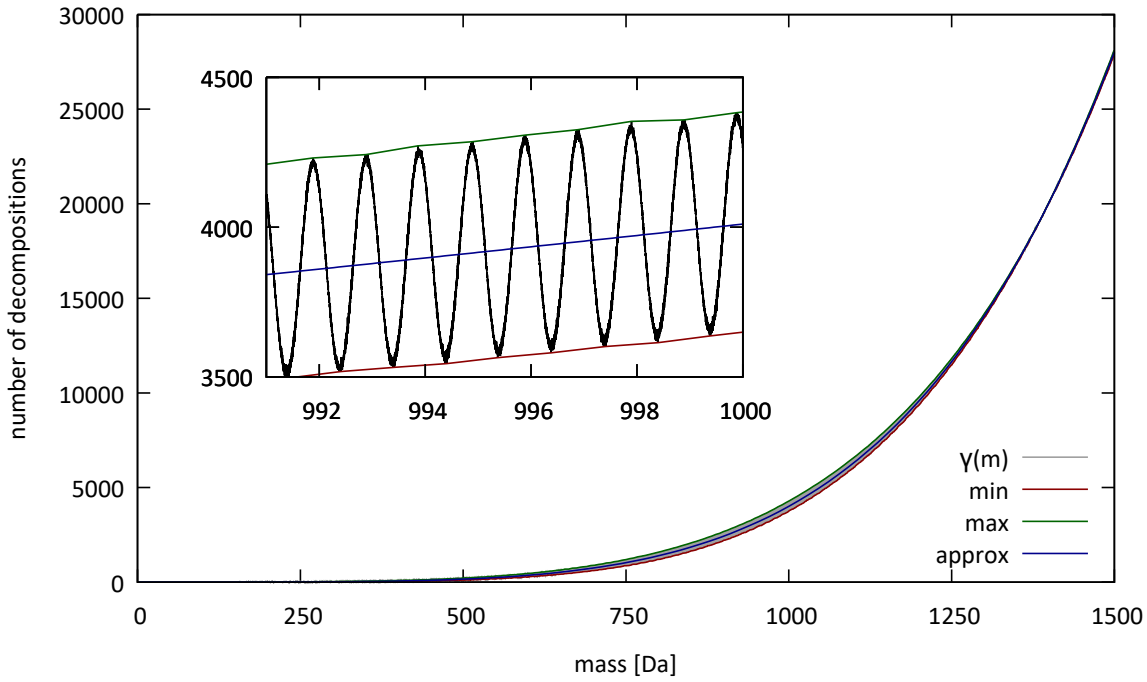


Figure 8.2: Number of decompositions over the alphabet of elements CHNOPS. Numbers for bins of width 0.001 Da, corresponding to absolute mass accuracy 0.0005 Da. True number of decompositions $\gamma(M)$ and approximation (8.8). The number of decompositions is varying periodically, so we use "superbins" of width one Dalton and display minimum and maximum in each superbin. As shown in the inset, $\gamma(M)$ varies with a periodic function of period roughly 1 Da.

we have measured a peak with mass M, then the number of decompositions in the interval $[M, M+\varepsilon]$ for mass accuracy $\frac{1}{2}\varepsilon$ is

$$\gamma(M,\varepsilon) \approx 3.10657 \cdot 10^{-9}\varepsilon M^5 + 8.22867 \cdot 10^{-7}\varepsilon M^4 + 8.05088 \cdot 10^{-5}\varepsilon M^3. \tag{8.8}$$

Empirically, we can simply count the number of decompositions over CHNOPS. We have plotted the number of decompositions over the alphabet of elements CHNOPS in Fig. 8.2. We observe a periodic variation of the number of decompositions; but unlike numbers for amino acids (see below), the variation is relatively small. In particular, the number of decompositions changes only slightly when we move from one bin to the next; furthermore, Eq. (8.8) is an accurate approximation of the true number. We attribute this "non-combinatorial behavior" (see Fig. 8.3) to the fact that hydrogen with mass almost one is present in the weighted alphabet. We have deliberately not filtered molecular formulas (Sec. 8.4) as this might again introduce unpleasant combinatorial effects.

The lesson to be learned from the above is simple: Even if we have an excellent MS instrument with mass accuracy 1 ppm or below (recall the difference between "anecdotal mass accuracy" [379] and everyday mass accuracy), the number of decompositions becomes large as masses exceed 1000 Da — even for the small alphabet of elements CHNOPS. This means that we cannot hope to recover the correct molecular formula from monoisotopic mass alone. In their 2006 paper "Metabolomic database annotations via query of elemental compositions: Mass accuracy is insufficient even at less than 1 ppm", Kind and Fiehn [189] also consider chemical restrictions and find — well, pretty much what the title suggests.

8.6 Decomposing amino acids

Let us consider the task of finding all amino acid compomers for a given mass. The first and potentially most important observation here, is that the number of decompositions will quickly explode. Even if we

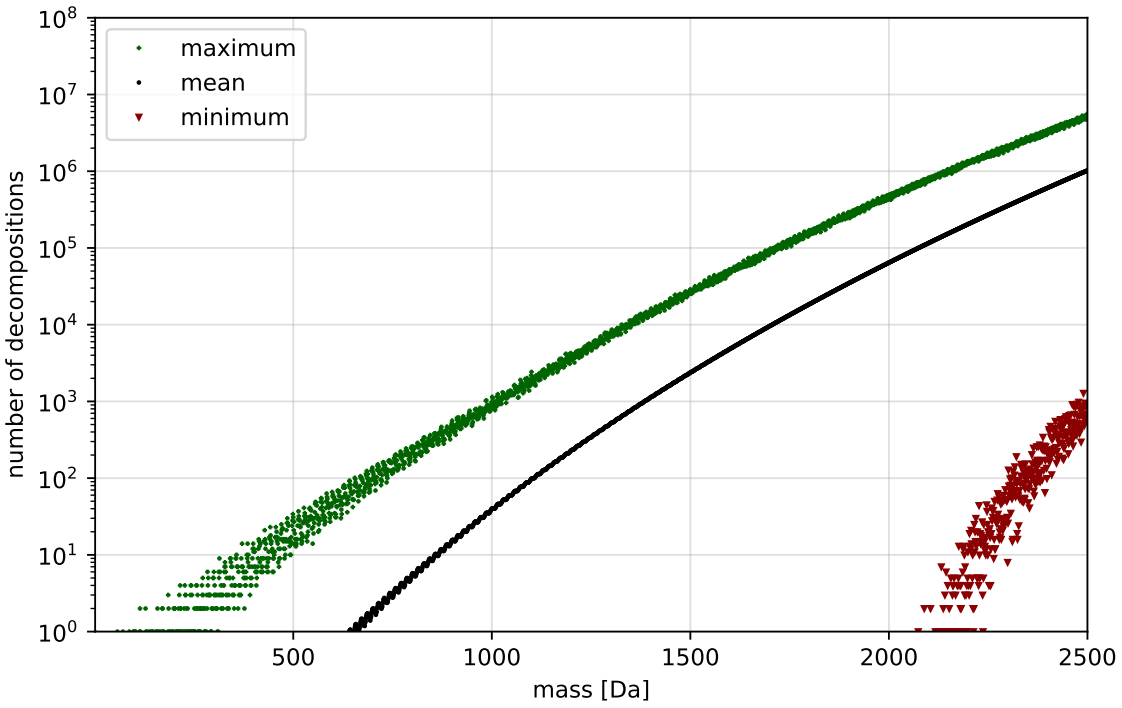


Figure 8.3: Number of amino acid decompositions with mass M. Leucine and isoleucine are treated as a single character. Numbers for bins of width 0.001 Da, corresponding to absolute mass accuracy 0.0005 Da. The number of decompositions is varying strongly, so we use "superbins" of width one Dalton, and only display maximum, mean, and minimum in each superbin. Note the logarithmic y-axis; values of zero are not shown in the plot.

have an instrument with ideal mass accuracy, we cannot tell apart certain compomers over the amino acid alphabet.[2] This goes beyond the obvious leucine/isoleucine ambiguity mentioned before: For example, $\mu'(\mathrm{AD}) = \mu'(\mathrm{EG})$, $\mu'(\mathrm{AG}) = \mu'(\mathrm{Q})$, and $\mu'(\mathrm{GG}) = \mu'(\mathrm{N})$ for amino acid residues alanine A, aspartic acid D, glutamic acid E, glycine G, asparagine N, and glutamine Q. These amino acid strings have *identical molecular formulas*, and so do the corresponding peptides (add H_2O). Consider mass 840.347442 that may be decomposed into three Q and four N: We can replace each Q by AG and each N by GG, resulting in 20 decompositions with identical molecular formula and, hence, identical mass. The problem becomes even more pronounced on "real-world" instruments which cannot have ideal mass accuracy.

We can also argue from an empirical perspective: Fig. 8.3 shows the number of decompositions over the amino acid alphabet, where leucine and isoleucine are treated as a single character. This chart was computed using recurrence (3.2) from page 43, and calculations were carried out with accuracy 10^{-3} Da: Real-valued masses were multiplied by 10^3, rounded, and used as integer masses for the recurrence. The number of decompositions varies strongly, and if we plotted the resulting numbers directly, you would be unable to see a thing in the figure.[3] Instead, we binned numbers of decompositions once more, this time into "superbins" of width 1 Dalton, each containing 1000 values; for each superbin, we report the minimum, mean, and maximum number of decompositions of any mass in the bin. Note the logarithmic scale for the y-axis: For very large masses, the number of decompositions can be approximated by a polynomial of degree 18, see Sec. 3.7; compare with the polynomial of degree 5 in (8.8) for CHNOPS. Still, the growth is sub-exponential, as can be seen in the figure: An exponential growth would correspond

[2] Remember that this is not true, as structure determines energy and, hence, mass; also remember that the mass accuracy required to distinguish structures with identical molecular formula, is far beyond what mass spectrometry will reach in the next decades.

[3] That is not entirely true; the resulting plot is in fact very interesting and even somewhat beautiful, see Exercise 8.6.

to a straight line. If we plotted these numbers using a regular scale, this would result in a rather boring plot: Compared to the largest numbers reported (several million), numbers below 10^5 would be almost indistinguishable from zero.

We can learn three things from Fig. 8.3: First, the number of decompositions is relatively small for masses below 1000 Dalton: at most about 1000 decompositions for *every* 0.001 Dalton bin. Second, the number of decompositions "explodes" beyond 2000 Da, where we already have to consider more than 60000 decompositions on average. This is mostly independent of the mass accuracy of the instrument. Third, combinatorial effects are still huge at 2500 Da, and the number of decompositions is practically impossible to approximate. In particular, there exist "sweet spots" beyond 1500 Dalton where we find no or few decompositions, although the average number of decompositions is already 10000 and higher.

In case you nevertheless want to decompose over the alphabet of amino acids, you might want to use a two-step approach: First, you decompose the molecular formula, which is relatively easy as the alphabet of elements is restricted to CHNOS. You can then identify the best molecular formula candidates using, say, the isotope pattern of the peptide. You should probably not restrict yourself to only one candidate. Third, you "decompose the molecular formula": Böcker and Pervukhin [35] show how we can modify our decomposition methods for this purpose. It should be noted that running time is not the limiting factor, as decompositions can usually be computed in less than 0.001 seconds for peptides below 2000 Dalton [35]. But the question is: What do we do with the resulting amino acid decompositions? If we are unlucky, there exist hundreds of amino acid compomers with identical molecular formula and identical isotope pattern; these decompositions correspond to an even bigger number of amino acid strings…

8.7 Decomposing average masses

We come back to the problem of generating molecular formula candidates; but this time, we assume that the monoisotopic peak is lost. This can happen if the molecular formula contains elements that are not frequent in biomolecules, see Table 7.5. For example, the amino acid selenocysteine $C_3H_7NO_2Se$ has a monoisotopic peak with relative intensity

$$0.9893^3 \cdot 0.99988^7 \cdot 0.99636 \cdot 0.99757^2 \cdot 0.0086 \approx 0.0082.$$

Clearly, this peak with relative intensity below 1 % is easily missed by the peak picking software. Other elements that may lead to a missing monoisotopic peak are tungsten, as well as boron, bromine, and iron if several such atoms are present. In contrast, if we stick with the classical elements CHNOPS then we can assume that the monoisotopic peak is present, unless we are considering proteins or very large peptides: You need 279 carbon atoms, 24963 hydrogen atoms, 822 nitrogen atoms, 1232 oxygen atoms, or 59 sulfur atoms so that the relative intensity of the monoisotopic peak is below 5 %. So, the resulting molecule has mass at least 1886 Da if it is made solely from sulfur, mass 3348 Da if it is solely made from carbon, and even higher mass for the other elements. The peptides of smallest mass where the monoisotopic peak drops below 5 %, consist of 34 cysteine residues (mass 3520.323 Da) or 27 methionine residues (mass 3556.104 Da), see Exercise 7.9 in the previous chapter.

For completeness, we now show how to determine the molecular formula of a molecule in case elements with "strange" natural isotope distributions are (potentially) present. For this, we will decompose the average mass of the molecule, corresponding to its atomic weight. Unfortunately, our inevitably incomplete knowledge of the natural isotope distribution leads to inaccurate knowledge of the atomic weight of molecular formulas. Consequently, decompositions must be executed with a huge mass inaccuracy of, say, 1000 ppm.

Given the observed normalized intensities $h_0, \ldots, h_{n_{\max}}$ and peak masses $M_0, \ldots, M_{n_{\max}}$ as the experimental isotope pattern of some query molecule. This allows us to estimate its *average mass* as $M_{\text{av}} := \sum_i h_i M_i$. This will underestimate the average mass as we are missing peaks in the isotope pattern with high mass but low intensity. The atomic weights of elements can be found in Tables 7.1 and 7.5; we decompose the number M_{av} over these weights. As noted, this will result in a huge number of decompositions. In

case running time is not an issue, this approach can nevertheless be executed and will result in mostly the same identification rates as decomposing the monoisotopic mass; we generate, say, 1000 times more candidates, but I claim that the 999 additional molecular formula candidates will have isotope patterns far, far different from the experimental isotope pattern, and not interfere with the identification of the correct molecular formula.

Now, let us assume that running time is an issue; can we again speed up the decomposition? Recall that we can decompose integers only, so we assume that all masses and other values are rounded using appropriate precisions. The mass of the monoisotopic peak is an *additive invariant* of the decompositions we are searching for: Given any solution, the sum of monoisotopic masses of all elements is the input mass; more precisely, it lies in some integer interval $l, \ldots, u$. The same holds true for the average mass. The interesting point is that there exist many more additive invariants in the isotope pattern, see Exercises 8.9 to 8.11; and, that knowing several of these additive invariants simultaneously makes the decomposition process significantly faster, as described by Böcker *et al.* [30].

8.8 Detour: Smarter rounding

The idea of using a blowup factor, as introduced in this chapter for mass decomposition, is also useful for other applications: Often, our algorithms assume that we have, say, unit masses so that we can do dynamic programming or other tricks. We do so, although measurements with high mass accuracy (10 ppm and below) are nowadays the rule rather than the exception. To get around this issue, we can simply store the high-precision mass of the peak of every 1 Da interval (bin) for scoring; as long as there is at most one peak in each 1 Dalton bin, this simplification will usually do the trick. (There may be multiple peaks in the experimental spectrum, in which case we simply use the maximum-scoring peak in the bin.) See, for example, Chapter 11.

Unfortunately, biomolecules contain many hydrogen atoms, and the monoisotopic mass of hydrogen is *larger* than its nominal mass; consequently, the monoisotopic mass of a biomolecule is usually larger than its nominal mass, and if we simply round the measured, high-accuracy mass, we will nevertheless end up in the wrong bin, that is, miss the correct nominal mass of the molecule. For example, a peptide with as few as 12 leucine residues (molecular formula $C_{72}H_{134}N_{12}O_{13}$) has monoisotopic nominal mass 1374 but monoisotopic mass 1375.019332. A notable exception are perfluorinated compounds such as PFAS (per- and polyfluoroalkyl substances), the "forever chemicals", which show a negative mass defect, and are easily detected as such exactly for this reason.

What is a sensible multiplicative correction factor to counter this effect? Considering peptides, if we assume a distribution of amino acids as in some protein database, the averagine approach of Senko *et al.* [320] can help us again: The average amino acid residue $C_{4.9384}H_{7.7583}N_{1.3577}O_{1.4773}S_{0.0417}$ from [320] has nominal monoisotopic mass 111 and monoisotopic mass 111.054306. Thus, a somewhat natural correction factor is $c := 111/111.054306 = 0.999511$. We multiply all measured masses by c to increase chances that the integer part of the monoisotopic mass and the nominal mass are identical.

This correction will reduce but cannot solve the problems introduced by rounding masses: For example, a peptide made from 14 leucine has molecular formula $C_{84}H_{156}N_{14}O_{15}$, nominal mass 1600, and mass 1601.187460. On the other hand, a peptide with 16 cysteine has molecular formula $C_{48}H_{82}N_{16}O_{17}S_{16}$, nominal mass 1666, and mass 1666.157524. The nominal mass difference between the peptides is 66, but the mass difference is only $64.970064 < 65$. So, whatever your correction factor is: By rounding, one of these peptides will land in the wrong bin. (Certain restrictions apply: I assume that you want to choose a "reasonable" correction factor close to 1.0.)

It must be understood that the correction factor depends on the application at hand; we will apply a different correction if we want to analyze, say, glycans (Chapter 11). Furthermore, the correction is most effective for macromolecules where each building block is made from numerous atoms (and, hence, possible molecular formulas are restricted) and which have masses beyond 1000 Da.

A similar technique was proposed by Kendrick [182] in 1963, normalizing masses of hydrocarbons to CH_2. This results in the *Kendrick mass scale* with unit 'Ke' and normalization factor $\frac{14\,\text{Ke}}{14.015650\,\text{Da}}$.

8.9 Historical notes and further reading

Our presentation in this chapter largely follows the paper of Böcker *et al.* [31]. See Dührkop *et al.* [92] on how to make real-valued decompositions fast(er) in practice; in fact, the methods presented here are up to 1000-fold faster than the naïve search tree or folded loops algorithm. Note that it does not make a difference if we are rounding masses up or down; Böcker *et al.* [31] round up whereas Dührkop *et al.* [92] round down. The only difference is whether we have to adjust the lower or the upper bound. We have chosen "rounding down" here, as this is the more "natural" way of rounding. The methods presented here are available as part of SIRIUS 4 [90].[4]

The pipeline in Fig. 8.1 (taken from Scheubert *et al.* [309]) is modified from Kind and Fiehn [189], where the authors proposed building an automated pipeline, similar to the one presented in this chapter. The idea of assigning molecular formulas to peak masses, dates back at least to the year 1965, when the "Artificial Intelligence" program DENDRAL was created for this task, see Sec. 10.7 below.

Sec. 8.3 proposes to use the "absolute plus relative noise" model from Sec. 4.8 for scoring intensities; Dührkop *et al.* [90] showed that this substantially improves molecular formula identification compared to other intensity noise models.

There exist other tools for decomposing isotope patterns, but mostly, these methods are part of commercial and proprietary software, and no details on the underlying methods have been published. Several programs are available to decompose masses over the amino acid alphabet; most programs rely on the naïve approach for decomposing masses as presented in Sec. 8.1 and have to somewhat arbitrarily (and rather aggressively) restrict the search space, say, by introducing upper bounds on the number of certain amino acids, to avoid the combinatorial explosion.

Rogers, Scheltema, Girolami and Breitling [300] propose a quite different approach to assign molecular formulas to metabolites: Instead of treating each measurement individually, they assign molecular formulas to a batch of metabolite masses. Here, they use the fact that metabolites are connected to each other via chemical transformations [41]. They then use Gibbs sampling to estimate posterior probabilities of the molecular formula assignments, given the data. Unfortunately, this approach suffers heavily from bad priors: Molecular formulas with many connections to other molecular formulas will be much preferred by the Gibbs sampling, see Chapter 12.

We have seen that it is not reasonable to decompose large masses over the amino acid alphabet: If you are lucky, you will hit a "sweet spot" where only relatively few amino acid decompositions are found. (According to Fig. 8.3, the last "sweet spots" should be found for peptide masses around 2200 Dalton.) In fact, the mass spectrometry literature contains many such "anecdotal" decompositions. But we are interested in computational methods that work for *all input data*, not just a few hand-selected peptides. As a successful example of a computational method that decomposes masses over the amino acid alphabet, we mention Bertsch *et al.* [20] who use such decompositions as part of a divide-and-conquer strategy.

As far as I know, nobody uses isotope patterns as part of peptide identification, which is a pity. Whereas we cannot deduce the amino acid composition from the isotope pattern, it can be highly informative for database search.

8.10 Exercises

8.1 Describe a naïve recursive algorithm to compute all decompositions c of real-valued masses in the interval $[l', u']$, see Algorithm 8.1.

[4] https://github.com/sirius-ms/sirius

8.2 Implement the algorithm from the previous exercise, as well as Algorithm 8.1. Given alphabet $\{H, C, N, O\}$ with masses $\{1.007825, 12.0, 14.003074, 15.994915\}$, compute all decompositions of mass 100 with inaccuracy 0.01. How many are there? Compute all decompositions of the mass 3000 with inaccuracy 0.0001. How many are there? Compare running times.

8.3 How do running times change if we throw in elements P and S with masses 30.973762 and 31.972071?

8.4 Similar to Exercise 3.21, it is easy to modify Algorithms 8.1 and 8.2, when upper and lower bounds for each character are given. Show how this can be done.

8.5 Consider the weighted alphabet of elements Σ = CHNOPS. Compute $\Delta(b)$ for blowup factor $b = 5963.4$.

8.6 Compute and plot the number of decompositions over the 19-letter amino acid residue alphabet using mass accuracy 0.001 Dalton. Plot the exact number of decompositions for each mass (1000 points per Dalton). Try a logarithmic y-scale.

★ 8.7 Give an algorithm that transforms a molecular formula into all amino acid compomers with this molecular formula.

8.8 Give an algorithm that transforms an amino acid compomer c into all peptide strings s with $\text{comp}(s) = c$.

8.9 For every element E, let p_E denote the probability that an isotope of this element is monoisotopic. Show that the negative log intensity of the monoisotopic peak is an additive invariant over the weighted alphabet with weights $-\log p_E$. Note that phosphorus has weight zero over this alphabet.

★ 8.10 Show that the relative intensity between the +1 peak and the monoisotopic peak is an additive invariant.

★ 8.11 Show that the weighted mass difference $\frac{h_1}{h_0}(M_1 - M_0)$ between the +1 peak and the monoisotopic peak is an additive invariant, where h_0, h_1 and M_0, M_1 denote intensities and masses of the monoisotopic and the +1 peak.

9 Fragmentation trees

"All models are wrong but some are useful." (George E. P. Box)

As mentioned in the introduction, there are biomolecules beyond DNA, RNA, proteins and peptides: The phenotype of an organism is strongly determined by the small chemical compounds contained in its cells. These compounds are called *metabolites*; their mass is typically below 1000 Da. Metabolites are the intermediates and products of metabolism, that is, chemical reactions that happen in living beings to maintain life. Biopolymers such as proteins, DNA, or glycans (see Chapter 11 below) are not considered metabolites, but their constituent monomers (amino acids, monosaccharides) are.

In the following, our task is to determine the molecular formula of a single small molecule. (As mentioned, we can also try to determine the molecular formula of a not-so-small molecule, but this makes little sense.) This small molecule may be a metabolite (Sec. 10.1); but it may also be a drug, a drug degradation product — our body tries to get rid of drugs as soon as they have entered the system — or another xenometabolite, or some synthetic compounds such as the aforementioned antifouling agents or fungicides, which are frequently found in nature and taken up by living beings. The previous chapters presented methods that allow us to determine the molecular formula using the isotope pattern of the molecule; here, we concentrate on its fragmentation (tandem MS) spectrum. In practice, best results are achieved by combining both approaches.

9.1 Naïve approach: Fragmentation bushes

The simplest way to model the fragmentation of a molecule, is to assume that all fragments directly resulted from the precursor ion via a single fragmentation reaction. This is a somewhat trivial observation but can already help us to identify the molecular formula of the precursor ion: We can limit ourselves to those molecular formulas that allow us to explain each peak in the fragmentation spectrum, meaning that we find a subformula with mass sufficiently close to the peak mass.

Obviously, this very naïve approach may fail as soon as there is a single noise peak in the spectrum, see Chapter 2. A more reasonable — but still naïve — approach is to count the number of peaks that a molecular formula candidate explains. Even better, we can use the scores from Secs. 4.7 and 4.9 to take into account peak intensities and mass deviations, see also Sec. 9.6 below. Unfortunately, this is the furthest we can get without including prior information, see Chapter 12.

In the nomenclature of the next section, the naïve model corresponds to a fragmentation bush: In graph theory, a tree where every node but the root is one step from the root is called a *star*; we will use the more botanical *bush*.

9.2 Formal definition of fragmentation trees

Let us assume that we are given the tandem mass spectrum of a metabolite, but we have no information about its structure. In our presentation, we assume that the molecular formula of the metabolite is known, as this makes it easier to understand the details. See Sec. 9.7 on how to use the presented method to determine the molecular formula.

The fragmentation process of small molecules through tandem MS is not completely understood; in principle, a small molecule can fragment at almost any chemical bond. But here, we are in an even worse situation: We do not have any information about the structure of the metabolite! As we will see in the next

chapter, searching in a (relatively small) database of molecular structures is already a hard problem, but *de novo* structural elucidation is even harder: First methods exist, but they recover the correct structure only for a minority of compounds.

But maybe, we do not have to identify the structure of the metabolite to do something useful with the fragmentation spectrum? Consider genome sequencing: After you have assembled the genome, nobody will expect that you explain every base of the genome, or describe the exact function of every gene; but at least, you have to *annotate* it with additional information (open reading frames, genes, introns/exons, micro-RNAs etc.) so that it is useful for other scientists. That is what we want to do here: We want to *annotate the fragmentation spectrum* with additional information.

In experiments, we see that fragments of the precursor are further fragmented when higher collision energies are applied. Ion trap instruments allow us to build such fragmentation cascades experimentally. To account for this multi-step fragmentation, we want to use a *fragmentation tree* to annotate the fragmentation spectrum: This rooted tree has molecular formulas as its node set, and is rooted in the precursor molecular formula. Each node molecular formula *explains* a peak in the fragmentation spectrum: That is, its mass is within the allowed mass deviation of the measured peak mass. We demand that every peak is explained by at most one node. Peaks in the spectrum which are not explained by the fragmentation tree, are noise and must be taken into account as such in our scoring. Nodes are connected by directed edges, constituting *losses*: Parts of the molecule break off but are not ionized, so that we cannot detect them in the subsequent MS step. A directed edge uv in the tree tells us that fragment v is a sub-fragment of fragment (or precursor ion) u: So, for each element, the molecular formula of v contains at most as many atoms as the candidate molecular formula of u. Here, $u - v \geq 0$ is the loss. Recall that we use uv as shorthand for the edge (u, v).

Strictly speaking, what we are considering here is an *arborescence* or an *arboreal* rooted tree, where all edges are pointing away from the root. Given that "arboreal" means "tree-like", this is a somewhat funny definition. In the following, we will assume that all our rooted trees are arboreal, with edges pointing away from the root.

Different fragmentation pathways may lead to fragments with identical molecular formula or even identical structure. Hence, we have oversimplified the problem: By restricting ourselves to fragmentation *trees*, we demand that each fragment in the fragmentation spectrum is generated by a single fragmentation pathway. But this is not a serious oversimplification: First, this situation is the exception rather than the rule. Second, we can argue that we are interested in the major fragmentation events that mainly occurred. Third, the most common exceptions to our assumption are "parallelograms" where loss A is followed by loss B, as well as the other way round. (For example, two groups fall off at different ends of the molecule.) In this case, the tree representation has to artificially decide whether the fragment which lost both A and B is a child of the fragment for loss A, or the fragment for loss B. But this is implicitly encoded in the tree and can be easily recovered from there.

We also demanded that every peak in the spectrum is explained by at most one node of the fragmentation tree. But in contrast to the previous paragraph, not much discussion is needed for this constraint: Fragmentation spectra of metabolites are usually very sparse — ten peaks with significant intensity is already a lot — and the number of "potential fragment molecular formulas" (all subformulas of the precursor molecular formula) is in general many orders of magnitude larger. Given that the vast majority of potential fragments is *not* detected in the fragmentation spectrum, it is *extremely unlikely* that some peak in the fragmentation spectrum corresponds to two fragments with different molecular formulas but almost identical mass.

Finally, we demanded that the fragmentation tree must only contain nodes that explain a peak in the fragmentation spectrum. This may be justified by parsimony, Occam's razor (William of Ockham, circa 1287–1347, scholastic philosopher and theologian) or however you want to call it; but the easiest explanation is that we want to annotate the fragmentation spectrum. Peaks that were not observed, require no annotation.

The above discussion tells us that the constraints we put on fragmentation trees are not too unrealistic. But the question is: Why did we introduce them, anyway? Why not allow that two fragmentation cascades

lead to the same fragment? Why not allow that two nodes of the fragmentation tree explain the same peak? The answer is "optimization": What we will do in the next section, is to search for a tree that *best* explains the fragmentation spectrum, for a reasonable score. If we drop the constraint that every fragment has only one parent, then it will be beneficial, under most reasonable scorings, to give any node in the resulting graph *many* parents, to collect as much score as possible. It is not unlikely that some fragment will be a child of all other fragments in the spectrum. But this is much worse than our constraint from above, and much further away from the chemical reality. Similarly, if we drop the constraint that each peak is explained by at most one node, then it will often be beneficial to include all explanations in the resulting graph; again, this is much worse than our constraint.

In case you are unhappy with this argument ("it is not true, but better than the alternative"), be advised that this is nothing special. One can easily argue that sequence alignments and phylogenetic trees are as far away from the biological truth, as we are from the chemical truth. But both sequence alignments and phylogenetic trees have been extremely helpful tools for the analysis and understanding of biology.

Formally, a directed graph $T = (V, E)$ is a *tree* if there is a root node $r \in V$ such that every node $v \in V$ can be reached from r via a unique path. (Recall that this is also called "arborescence".) Many alternative characterizations of trees exist; we just mention that T is a tree with root r if and only if every node $v \in V$ can be reached from r, and every node v but the root has in-degree one, whereas the root has in-degree zero. Here, "v can be reached from r" means that there is a directed path from r to v; this, in turn, means a path $p = v_0 v_1 \dots v_l$ with $v_0 = r$, $v_l = v$, and $v_{i-1}v_i \in E$ for $i = 1, \dots, l$. Clearly, any tree is acyclic: Recall that a graph G is acyclic if, for every node v of G, there is no directed path from v to v in G. A tree $T = (V, E)$ is a *fragmentation tree* if all nodes $v \in V$ are molecular formulas, and an edge $uv \in E$ implies that v is a subformula of u. A fragmentation tree *explains* some fragmentation spectrum if the root $r \in V$ is the molecular formula of the precursor peak; for every $v \in V$ there exists some peak in the fragmentation spectrum with mass sufficiently close to $\mu(v)$; and no two nodes map to the same peak in the fragmentation spectrum.

9.3 Fragmentation graphs and the Maximum Colorful Subtree problem

How can we find the fragmentation tree that *best* explains the fragmentation spectrum? To do so, we transform the fragmentation spectrum into a "fragmentation graph". We then show — in fact, there is not much to be shown — that every fragmentation tree that explains the fragmentation spectrum must also be a subtree of the fragmentation graph, and *vice versa*.

Our first step is to transform the masses of fragment peaks into molecular formulas, which will serve as the nodes of our *fragmentation graph* $G = (V, E)$. Since all fragments must originate from the precursor ion, we have to consider only subformulas of the (known) molecular formula of the precursor ion. As always, we only consider molecular formulas which are sufficiently close to the peak mass. Methods for doing so were discussed in Chapter 3 and Sec. 8.1. Clearly, there can be more than one decomposition of each fragment mass. These molecular formulas plus the precursor molecular formula are nodes of G. The graph is node-colored: Every node gets a color that represents the peak it explains. For edges, we create a directed edge between two nodes if one molecular formula is a subformula of the other molecular formula. Doing so, we represent *every possible fragmentation step*, even those that are extremely unlikely.

The resulting fragmentation graph $G = (V, E)$ is a directed, acyclic graph (DAG). Recall that *acyclic* means that we cannot walk away from some node v of the graph along directed edges, and ultimately end up in v again. Since the "subformula" relation is transitive, the constructed graph is also transitive: $uv, vw \in E$ implies $uw \in E$.

We claim that every fragmentation tree that explains the fragmentation spectrum must also be a subtree of the fragmentation graph: If a tree $T = (V_T, E_T)$ explains the fragmentation spectrum, then for every $v \in V_T$ there exists some peak in the fragmentation spectrum with mass sufficiently close to

$\mu(v)$; hence, the molecular formula v is also a node of the fragmentation graph. The definition of edges is identical for fragmentation trees and fragmentation graphs. Analogously, any subtree rooted in the root of the fragmentation graph, is a fragmentation tree that explains the fragmentation spectrum. Therefore, we will only consider colorful subtrees of the fragmentation graph in the following.

To formulate our task as an optimization problem, we still need some scoring to evaluate how well some fragmentation tree explains the fragmentation spectrum: For example, the tree which consists of one node only, corresponding to the precursor ion molecular formula, explains the fragmentation spectrum regardless of all other peaks. Clearly, this makes sense, as all other peaks may be noise; this is unlikely, but we cannot exclude the possibility. Nevertheless, we now want to find the best fragmentation tree — or, equivalently, the best subtree of the fragmentation graph. Usually, our first idea is to use a peak counting score: We count the number of peaks that can be explained as fragments of the precursor ion. But here, this will not take us very far: Counting peaks, the fragmentation tree where every node is connected to the precursor peak node (the "fragmentation bushes" from the previous section) will receive maximum weight; in addition, there will be numerous other fragmentation trees with identical score. To achieve good results, we have to use a more involved scoring.

Unfortunately, finding a reasonable scoring is non-trivial and requires prior knowledge; we give some details in Sec. 9.6. For the moment, we simply assume that we are given some edge weights via function $w : E \to \mathbb{R}$. We stress that edge weights may be negative: It may be more likely that a peak is noise than the explanation via a node of the fragmentation tree; but we may nevertheless choose to include it into the tree, as this allows us to better explain subsequent fragmentation reactions.

We now formalize the problem of computing a fragmentation tree. All of the following definitions are standard in computational graph theory, see Sec. 15.4: Let $n := |V|$ be the number of nodes and $m := |E|$ the number of edges in the graph, and let $k := |C|$ be the number of colors. Let $c : V \to C$ be the node colors of G. Furthermore, let $w : E \to \mathbb{R}$ be the edge weights, whose sum we want to maximize. We will write $w(u,v)$ instead of $w((u,v))$ or $w(uv)$ for an edge uv. A *subtree* $T = (V_T, E_T)$ of G is a subgraph of G, $V_T \subseteq V$ and $E_T \subseteq E$, that is a tree. One peak may result in many molecular formulas in V explaining it; to avoid that two nodes explain the same peak, we have to ensure that the subtree does not use any color twice. The tree T is *colorful* if it uses every color in C at most once: So, $c(u) \neq c(v)$ holds for all $u, v \in V_T$ with $u \neq v$. We formalize our problem as:

Maximum Colorful Subtree problem. Given a node-colored DAG $G = (V, E)$ with weights $w : E \to \mathbb{R}$. Find the colorful subtree $T = (V_T, E_T)$ of G of maximum weight $w(T) := \sum_{e \in E_T} w(e)$.

Unfortunately, this problem is NP-hard. Hence, we cannot compute a maximum colorful subtree in polynomial time, unless P = NP.[1] To make the consequences clear: We cannot hope to find an algorithm with running time, say, $O(n^{1000})$, where n is the number of nodes of the graph, unless P = NP. The problem remains NP-hard if we assume that all edges have positive or even unit weight, and if we remove the "colorful" constraint: That is, all nodes in our input graph have different colors or, equivalently, no colors at all. In contrast, we can easily compute a maximum subtree if we drop the requirement that it has to be colorful and all edges have non-negative weight; in this case, every node reachable from r simply chooses its incoming edge of maximum weight; as G is acyclic, this results in a maximum spanning arborescence. But it must be understood that limiting ourselves to colorful subtrees is inevitable, and so are negative edge weights, see Exercises 9.2 and 9.3. We mention in passing that the Maximum Colorful Subtree problem is closely related to the edge-weighted Graph Motif problem introduced in Sec. 10.5. This is somewhat surprising, as we are working with molecular graphs there, whereas it is fragmentation graphs now.

[1] The P versus NP problem is an unsolved problem in computer science, and one of the seven Millennium Prize Problems where you are awarded US$ 1,000,000 for the first correct solution. An answer to the P = NP question would determine whether problems that can be verified in polynomial time can also be solved in polynomial time. If P = NP holds, a large portion of the complexity hierarchy — something theoretical computer scientists have worked on for many decades — would collapse. It is under discussion how much impact this would have in practice.

9.4 Exact algorithms for the Maximum Colorful Subtree problem

Whereas the computational hardness of the Maximum Colorful Subtree problem may be daunting at first, this does not mean that we have to abandon all hope! In particular, we should not abandon our hope to find an exact solution to the problem; just because it is NP-hard, we do not have to fall back to heuristics for its solutions. In the following, we assume that all colors C are in use, so $c(V) = C$ where $G = (V, E)$.

A naïve algorithm for the problem is as follows: Let us demand that the fragmentation tree uses *exactly* nodes $U \subseteq V$ from the fragmentation graph. Then, we can compute the maximum subtree by letting every node in U except the root choose its best incoming edge from U. Iterating over all $U \subseteq V$ allows us to find the maximum colorful subtree: We do not know what the node set of the maximum colorful subtree *is*, but it has to have *one*; iterating over *all* subsets we cannot miss it. This results in running time $O(2^n \cdot m)$. Clearly, subsets $U \subseteq V$ must be colorful, meaning that $c(u) \neq c(v)$ must hold for all $u, v \in U$ with $u \neq v$. How many colorful subsets $U \subseteq V$ exist? For a color $c' \in C$ let $n(c') := \#\{v \in V : c(v) = c'\}$ be the number of nodes in G of that color. It is easy to see that the number of colorful subsets equals $\prod_{c' \in C}\big(n(c')+1\big)$; we have to add one as U does not have to use color c' at all. Unfortunately, no useful bound on this number is possible, and it can get large even for relatively small graphs: For a graph with 90 nodes and ten colors, such that every color is used by nine nodes, we have 10^{10} colorful subsets. But we have already found an algorithm that can solve such instances — which are far from what you want to try on a blackboard — exactly and in a matter of minutes.

9.4.1 Dynamic programming and fixed-parameter algorithmics

A faster algorithm for the problem uses dynamic programming: Let $D[u, S]$ be the maximum weight of a colorful subtree in G that is rooted in $u \in V$ and uses *at most* the colors from the color subset $S \subseteq C$. This clearly includes the case that it uses *all* colors from S. If there is no such tree, we assume $D[u, S] = -\infty$; this is the case if $c(u) \notin S$. Now,

$$D[u,S] = \max \begin{cases} \max\limits_{\substack{v \in V \text{ with } uv \in E,\\ c(v) \in S \setminus \{c(u)\}}} D\big[v, S \setminus \{c(u)\}\big] + w(u,v) \\ \max\limits_{\substack{(S_1,S_2) \text{ with } S_1 \cup S_2 = S,\\ S_1 \cap S_2 = \{c(u)\},\ S_1, S_2 \neq \{c(u)\}}} D[u,S_1] + D[u,S_2] \\ 0 \end{cases} \tag{9.1}$$

with initial condition $D\big[u, \{c(u)\}\big] = 0$ for all $u \in V$. The third case covers the tree consisting of u alone, which uses at most the colors from S. Calculations can be carried out by iterating over the nodes u in reverse topological order; for computing fragmentation trees, this simply means that we go from smaller masses to larger masses. When the array has been filled, the optimal weight of a colorful subtree is $D[r, C]$ where r is the root of the DAG. The tree can now be recovered from the matrix by backtracing, see Chapter 2 and Sec. 15.5.

The first line of recurrence (9.1) extends a tree by introducing u as the new root, adding the weight of the edge uv to the weight of the tree below v. The second line merges two trees which have nothing in common but their root u; S_1, S_2 form a partition of S except for the color $c(u)$ of u. Now, an optimal subtree rooted in u and using at most the colors from S must have either no edge leaving u (third line), exactly one edge leaving u (first line), or two or more edges leaving u (second line). This shows that recurrence (9.1) is correct.

Clearly, the second line of (9.1) is the expensive part of the calculation, as we have to iterate over all subsets $S_1 \subseteq S$ with $c(u) \in S_1$. Also clearly, we need $O(2^k \cdot n)$ space to store the array D. For running time, a naïve analysis would lead to an upper bound of $O(4^k \cdot km)$, as we have to iterate over $O(2^k)$ subsets $S \subseteq C$ and then over $O(2^k)$ subsubsets $S_1 \subseteq S$. But a closer analysis reveals that the algorithm needs only $O(3^k \cdot kn)$ time to calculate the second line of the recurrence. This is because C is actually partitioned into *three* subsets: These subsets are $C - S$, S_1 and S_2. (The fact that $S_1 \cap S_2 = \{c(u)\}$, is not relevant for our

considerations as it introduces only a constant factor.) Clearly, there are 3^k possibilities to perform this partitioning. As the second line of recurrence (9.1) dominates the running time, we reach total running time $O(2^k \cdot m + 3^k \cdot kn)$, where the factor k accounts for set operations.

Running time and space of the dynamic programming algorithm are polynomial in the number of nodes and edges, despite the computational complexity of the problem. The exponential growth of space and running time is restricted to the number of colors k, being the number of peaks in the fragmentation spectrum. Algorithms with running time $O(f(k) \cdot n^\beta)$ where f is a computable function, k is a *parameter* describing the problem instance, n is the problem size and β is a constant, are called *fixed-parameter algorithms*, and a problem that allows for such an algorithm is called *fixed-parameter tractable* with respect to parameter k. I stress that this is much stronger than the statement, "running time is polynomial for any fixed k" — for a fixed-parameter algorithm, the degree of the polynomial must not change when k increases. Unlike other algorithmic concepts, fixed-parameter algorithms are of practical use in bioinformatics, as they are indeed swift as long as the parameter does not get too large. We see that the algorithm presented above *is* a fixed-parameter algorithm for the Maximum Colorful Subtree problem, and the problem is fixed-parameter tractable with respect to parameter "number of colors in the graph".

As an algorithm engineering trick, we can define $D[u, S]$ to be the maximum weight of a colorful subtree in G that is rooted in $u \in V$ and uses *exactly* the colors from $S \subseteq C$. Again, we assume $D[u, S] = -\infty$ if no such tree exists. But unlike above, $D[u, S] = -\infty$ now holds for the majority of the array D. Again, we initialize $D\big[u, \{c(u)\}\big] = 0$ for $u \in V$, and compute D using recurrence (9.1). In the end, the optimum weight can be found as $\max_{S \subseteq C} D[r, S]$ for root r. To save space, we do not store the complete matrix but only those entries that are different from $-\infty$; this can be achieved using a hash map. To save time, we have to transform recurrence (9.1) into its bottom-up variant, see Sec. 15.5. Unfortunately, all of this does not change the asymptotic bounds for time and space.

9.4.2 Integer linear programming

The second exact algorithm for the Maximum Colorful Subtree problem uses Integer Linear Programming (ILP). But before we can define what an integer linear program is, we have to define what a linear program is: Much like "Dynamic Programming", the word "program" in "Linear Program" does not mean what it means today, see Sec. 15.5.

In short, a linear program is an optimization problem where both the objective function and the constraints are linear. We are searching for a vector $x \in \mathbb{R}^n$ in n-dimensional space. As our first constraint, we demand that each coordinate of $x = (x_1, \ldots, x_n)$ is non-negative, $x_i \geq 0$. We will write this as $x \geq 0$; compare with "$y = z$" for vectors y, z which also means that each coordinate is equal. We want to maximize the linear objective function $d^\mathrm{T} \cdot x = d_1 x_1 + \cdots + d_n x_n$ where $d \in \mathbb{R}^n$ is some fixed vector. (Vector "d" is called "c" in the literature, but that letter is already in use here for the color mapping.) Negative coordinates are allowed for d; if you prefer to minimize, use $-d$. We have to transpose vector d (flip it from a column vector to a row vector) because both d and x are column vectors.

This would still be a rather boring optimization problem; as long as d has one positive coordinate, we can make $d^\mathrm{T} \cdot x$ arbitrarily large. To this end, we introduce further constraints: For $i = 1, \ldots, m$ we demand that $a_i^\mathrm{T} \cdot x \geq b_i$ where $a_i \in \mathbb{R}^n$ is a vector and $b_i \in \mathbb{R}$ is a scalar. A more elegant way to write this is $Ax \geq b$ (see above) where the a_i^T are the rows of the $m \times n$ matrix A and $b = (b_1, \ldots, b_m)$. See Fig. 9.1 for an example.

Numerous optimization problems can be formulated as a linear program (LP), such as problems from operations research: How much of which item do I have to produce so that my revenue is maximized? See Exercise 9.6. The reason why linear programming has been so tremendously successful, is that there exist numerous free and commercial solvers that allow us to solve huge instances in little time: Instances with millions of variables and millions of constraints can be solved in reasonable time, as long as the matrix A is reasonably sparse (does not contain too many non-zero entries). On the theoretical side, linear programs can be solved in (weakly) polynomial time using interior point methods.

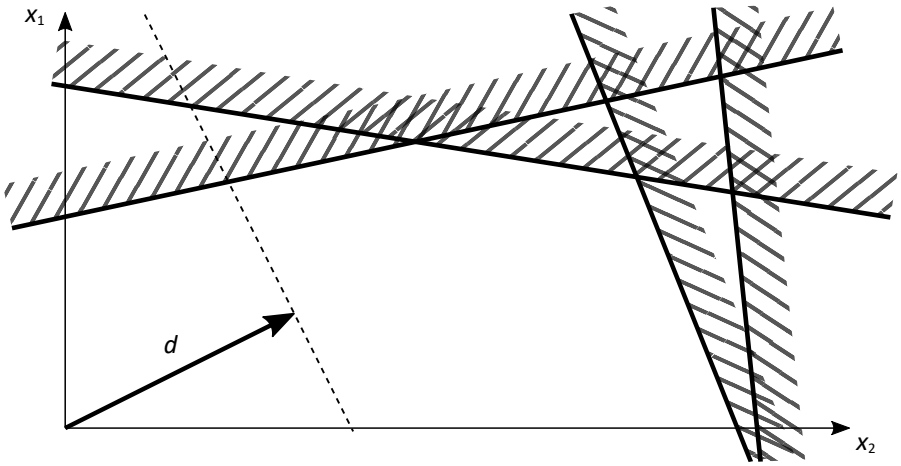


Figure 9.1: Illustration of a linear program with two variables $x = (x_1, x_2)$. The vector d shows the direction in which the objective function increases; the orthogonal line, where all solutions have equal objective function, is shown as dashed. Several constraints are indicated as lines; the forbidden half-plane is indicated by stripes.

Integer linear programming is basically identical to linear programming, except for one important difference: We demand that entries in the solution vector x are integer, $x \in \mathbb{Z}^n$. With this small change, the complexity of the problem changes to NP-hard. This is not much of a surprise: Numerous combinatorial problems can be easily encoded (with polynomial blowup) as an Integer Linear Program (ILP), including numerous NP-hard problems and, in particular, the Maximum Colorful Subtree problem. If we were able to solve any ILP in polynomial time, then we could find a solution of the Maximum Colorful Subtree problem in polynomial time — and we already know that this problem is NP-hard.

So, why care? The answer is "solvers": Numerous free and commercial solvers exist that can, despite the theoretical hardness of the problem, find exact solutions to relatively large ILP instances in reasonable time. (I deliberately stay vague here, because it depends on the structure of the problem we are encoding how large instances may become before running times get prohibitive.) It is understood that solving an ILP usually requires significantly more time — often, many orders of magnitude — than solving an LP. Algorithm engineering techniques such as facet-defining inequalities, "branch and cut", (delayed) column generation or "branch and price" have been developed to solve large ILP instances from particular combinatorial problems even faster. These techniques are based on the fact that we can quickly solve an LP, and only by demanding that variables are integer, we make things complicated. From personal experience, I can say that it can be highly challenging to beat an ILP using other algorithmic techniques such as fixed-parameter algorithms.

Back to the Maximum Colorful Subtree problem: We search for a colorful subtree in the input graph of maximum weight. It is understood that any subgraph of a directed, acyclic graph G is again a directed, acyclic graph. But when is a tree a tree?[2] Remember that a directed graph T is a tree with root r if and only if every node $v \in V$ can be reached from r, and every node v but the root has in-degree one, whereas the root has in-degree zero. For the ILP, we use a slightly different characterization: A directed, acyclic graph T is a tree if and only if every node v but the root has in-degree one, whereas the root has in-degree zero; see Exercise 9.5. This is intuitively easy to understand: If you want to leave a node of the tree, you must have entered it first; but if you have two ways of entering a node, then there exist two edge-disjoint paths between two nodes and, hence, the graph cannot be a tree.

We are given a node-colored DAG $G = (V, E)$ with weights $w : E \to \mathbb{R}$; we want to find the colorful subtree $T = (V_T, E_T)$ of G of maximum weight. We encode this as an ILP as follows: For each edge $e \in E$ of the graph, we generate one variable x_e that decides whether e is part of the subgraph ($x_e = 1$) or not

[2] This should read "When is a subgraph a tree?", but the other variant is more fun.

($x_e = 0$). In the following, we may assume that edges have been numbered $1, \dots, m$, so $x = (x_1, \dots, x_m)$. Any ILP demands that $x \geq 0$; to ensure that coordinates are zero or one, we simply add the constraint $x \leq 1$. We set $d_e := w(e)$, then $d^{\mathrm{T}} \cdot x$ is the weight of the selected subgraph.

At this point, the optimal solution would simply be to select ($x_e = 1$) all edges of G with positive weight; we have to enforce that only subtrees can be selected. From our characterization of trees, we know that this is the case if every node of the tree but the root has *exactly* one incoming edge; this means that every node of the tree but the root has *at least* one incoming edge and *at most* one incoming edge. (As noted above, we will enforce that the root of G is also the root of the subtree; but as noted in Sec. 9.8 below, this does not change the problem fundamentally, and is in sync with our application of finding a fragmentation tree.) For the first requirement, $x_{uv} = 1$ means that both u and v are part of the subgraph; clearly, v has (at least) one incoming edge. To enforce that u has an incoming edge, too, at least one edge that enters u has to be part of the subgraph. Let

$$E^-(u) := \{(w, u) \,:\, w \in V, wu \in E\}$$

be the set of edges in G entering u. If there exists at least one edge entering u then $\sum_{e \in E^-(u)} x_e \geq 1$; so we encode this requirement as a constraint

$$\sum_{e \in E^-(u)} x_e \geq x_{uv} \quad \text{for all } uv \in E, u \neq r \tag{9.2}$$

where r is the root of G. For the second requirement, any node v must have at most one incoming edge; this can be enforced via the constraint

$$\sum_{e \in E^-(v)} x_e \leq 1 \quad \text{for all } v \in V. \tag{9.3}$$

With these constraints, any subgraph encoded by edges e with $x_e = 1$ is indeed a tree.

So, we have enforced that the subgraph is a tree — or, more precisely, that only trees are taken into account when maximizing the ILP; but what about the requirement that the subtree has to be colorful? For a color $c' \in C$ let $V(c') := \{v \in V \,:\, c(v) = c'\}$ be the set of nodes that have this color. Our last problem is that "colorful" is a statement about the nodes of the subtree, whereas we are concentrating on edges in our ILP formulation. But luckily, we already know that every node of the tree, except for the root, has exactly one incoming edge! Hence, for any node $v \neq r$ in the tree there exists exactly one edge $e = uv \in E$ with $x_e = 1$. So, we can change the constraint to: For every color, there must be at most one edge entering a node of this color. Formally, let

$$E^-(U) := \bigcup\nolimits_{u \in U} E^-(u) = \{(w, u) \,:\, u \in U, wu \in E\}$$

be the set of edges entering a subset $U \subseteq V$ of nodes; then,

$$\sum_{e \in E^-(V(c'))} x_e \leq 1 \quad \text{for all } c' \in C. \tag{9.4}$$

But if we demand that (9.4) is satisfied, then (9.3) is automatically satisfied, too! If there is at most one edge entering *all* nodes with a particular color, then there is also at most one edge entering *any* node of that color; and we demand that this holds for every color. (The root is always part of the tree, so its color is used, and we may assume that the graph does not contain any other nodes of this color.) Therefore, we can leave out constraint (9.3) from our ILP, without changing the solution.

In full, our ILP can be formulated as follows:

$$\begin{aligned} \max \quad & \sum_{e \in E} w(e)\, x_e \\ \text{s. t.} \quad & \sum_{e \in E^-(V(c'))} 1 \cdot x_e \leq 1 && \forall c' \in C \\ & \sum_{e \in E^-(u)} (-1) \cdot x_e + 1 \cdot x_{uv} \leq 0 && \forall uv \in E,\ u \neq r \\ & x_e \in \{0, 1\} && \forall e \in E \end{aligned} \tag{9.5}$$

I have slightly changed the constraints to make them look similar to a normal form $Ax \leq b$ (equivalent to the form $Ax \geq b$ from above, as $\alpha \leq \beta$ is equivalent to $-\alpha \geq -\beta$). All entries in the resulting matrix A are -1, 0, or $+1$; all entries in b are 0 or 1. Furthermore, the matrix A is sparse, meaning that only a few entries are non-zero for non-degenerate instances of the Maximum Colorful Subtree problem. Recall that $x_e \in \{0,1\}$ is equivalent to ($x_e \geq 0$, $x_e \leq 1$, and x_e integer). The universal quantifier "$\forall$" means "for all" and is rarely used outside of mathematical logic — and (integer) linear programming.

So, when we are given an instance of the Maximum Colorful Subtree problem, we proceed as follows: We create the matrix A and the vectors b, d as implicitly described in (9.5). We then call an ILP solver with the resulting ILP instance. Finally, we extract all $e \in E$ with $x_e = 1$ — et voilà, this is the maximum colorful subtree.

9.5 Heuristic algorithms for the Maximum Colorful Subtree problem

Now that we have several exact algorithms at hand, we can think about heuristics for the problem. I argue against first developing heuristics: You will not even notice if your heuristics are doing hilariously stupid things, unless you can evaluate them against exact solutions, at least for some reasonably small instances.

Note that there exist at least three ways to evaluate the quality of a heuristic: The first is the simplest, comparing the objective value of the heuristic solution with that of the exact solution. This is a good way to evaluate solutions if you want to minimize the length of a round trip through Norway, or maximize revenue by placing stores in Vietnam. In such cases, the objective value has a real-world explanation (time, money) and we can judge if we accept that the heuristic solution is, say, 1 % off the optimum. But in bioinformatics, this is rarely the case; the objective function is of little or no interest by itself and usually not even reported to the user. (We often use the objective function value of a solution to judge its quality, though; but Chapters 5 and 6 have shown us that this is a non-trivial undertaking.) The second is the hardest, where we compare the structure of the heuristic solution with that of the exact algorithm. But it is often non-trivial how this comparison has to be carried out, and how we can transform structural similarity into a numerical value. If we cannot even decide upon the "right" similarity measure, how can we then use it to evaluate heuristics? The third is an intermediate: We use the heuristic and the exact method to rank certain candidates, in our case molecular formulas for the precursor ion, and compare the ranking performance. Given that this is the task we formulated at the beginning of this chapter, this appears to be a reasonable evaluation criterion for the Maximum Colorful Subtree problem.

Numerous heuristics have been developed for the Maximum Colorful Subtree problem [36, 91, 286]; unfortunately, most of them are in fact doing "hilariously stupid things", at least if you compare their results with the corresponding exact solutions [91]. Furthermore, the description of heuristics is only rarely satisfying from an algorithmic point of view: What works and what does not is highly dependent on the structure of the problem and the real-world instances, and is often of little use for other problems. Nevertheless, I will now describe two simple and one slightly advanced heuristic, so that you can get an idea. All of them perform reasonably well for identifying the molecular formula of a small molecule.

The *Kruskal-style heuristic* is inspired by Kruskal's algorithm for computing minimum spanning trees [194]. We sort all edges of the graph by decreasing edge weight, then iteratively add edges from the sorted list, ensuring that the growing subgraph is colorful and that each node has at most one incoming edge. Since r is the unique source of G, and since G is transitive, this will ultimately result in a colorful subtree of G. With regard to running time, sorting all edges according to weight takes $O(m \log n)$ time. Connectivity testing can be performed in sub-logarithmic time per considered edge using a union-find data structure [343]; checking for colorfulness is easily accommodated by initially placing all nodes of the same color in the same component. The overall time complexity is $O(m \log n)$.

The *Prim-style heuristic* is inspired by Prim's algorithm (previously developed by Vojtěch Jarník) for computing a minimum spanning tree [275]. The tree $T = (V_T, E_T)$ initially contains only the root r of G. In every step, we consider all edges uv with $u \in V_T$ and $v \notin V_T$ such that $c(v) \notin c(V_T)$; among these, we choose the edge with maximum weight and add it to the tree. We repeat until all colors in the graph

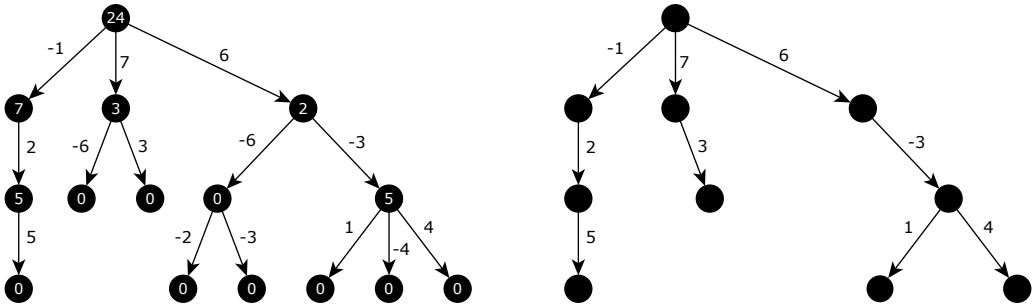


Figure 9.2: Illustration of the *Remove Dangling Subtrees* postprocessing. Left: Input tree, where each node v is labeled by its score $D[v]$. Right: Output tree of weight 24. Figure from [91].

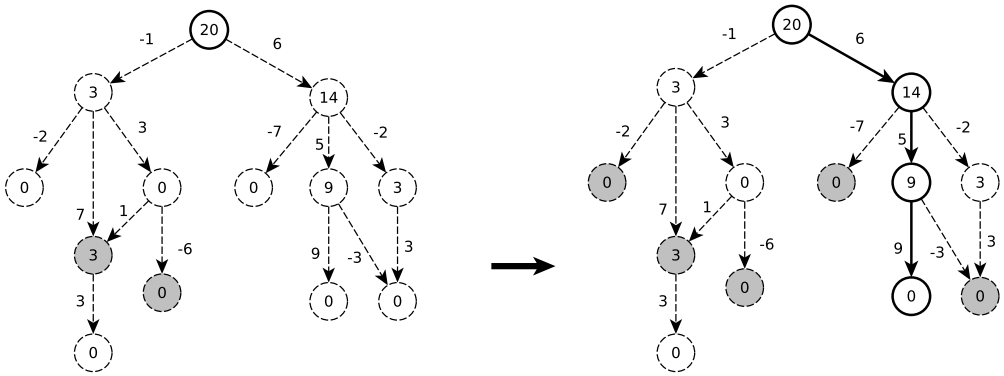


Figure 9.3: Example of the *Critical Path* heuristic. Nodes are labeled by score, solid lines show the tree, dashed lines the rest of the graph. Grayed-out nodes have colors already used in the subtree. Figure from [91].

are used in the tree; recall that G is transitive, so $rv \in E$ for each $v \neq r$. We explicitly do not quit when adding the first negative-weight edge uv, as the newly reached node v may allow us to later add other edges with positive weight. The Prim-style heuristic will usually result in a different tree than the Kruskal-style heuristic, due to the colorfulness constraint. The Prim-style heuristic requires $O(m \log n)$ time, which can be seen analogously as for the Kruskal-style heuristic.

Unfortunately, both algorithms will add edges even if only negative edge weights are left in the list. For this reason, we apply a postprocessing called *Remove Dangling Subtrees*: Let $T = (V_T, E_T)$ be the colorful tree we have computed by one of the two heuristics. For each node $u \in V_T$, let $D[u]$ be the maximum weight of any subtree rooted in u. Clearly, $D[u] \geq 0$, as the tree consisting solely of node u does have weight zero. For each edge uv of T with $w(u,v) + D[v] < 0$, we remove uv and the subtree below it, thereby increasing the weight of the tree. Table D can be computed using dynamic programming:

$$D[u] := \sum_{uv \in E_T} \max\{0, w(u,v) + D[v]\} \tag{9.6}$$

For that, we use a tree traversal, moving down and up every edge of T exactly once. The postprocessing requires $O(n)$ time, as every edge is considered once and $|E_T| = |V_T| - 1 < n$. See Fig. 9.2 for an example postprocessing.

Finally, here is one slightly more complex heuristic: The *Critical Path heuristic* searches for paths which should be added next to a growing tree. Again, we iteratively build a tree $T = (V_T, E_T)$; initially, the partial solution T contains only the root r of G. The score $S[u]$ of a node $u \in V$ is the maximum weight of a path p from u to any node v, such that $c(p) \cap c(V_T) \subseteq \{c(u)\}$; that is, the path does not use nodes with

colors already present in the tree, except for the color of the starting node. We can compute $S[u]$ using the recurrence

$$S[u] := \max_{uv \in E, c(v) \notin c(V_T)} \{0,\ S[v] + w(u,v)\}. \tag{9.7}$$

(We assume $\max \emptyset = 0$.) Recall that $c(V_T) = \{c(v) : v \in V_T\}$. The recurrence is correct because the coloring of G is *order-preserving*: That is, the colors of G can be ordered such that, for any edge uv, $c(u)$ comes before $c(v)$. This is not surprising; simply order the colors by the mass they represent. Hence, no two nodes of the path encoded by S have the same color. We iterate over the ordered colors c in reverse order, computing $S[u]$ for all nodes u of color c. The *critical path* p of maximum score can be found by backtracing from the maximum entry $S[u]$ with $u \in V_T$. We add p to T, then iterate, recomputing S for the new set of used colors $c(V_T)$. With regard to running time, we need $O(m)$ time to compute the $S[u]$ values and to identify the path of maximum weight. This is repeated at most $k-1$ times, resulting in a total running time of $O(km)$. See Fig. 9.3 for an example.

Beyond the designated heuristics presented above, it is noteworthy that exact methods for a problem almost always allow us to derive heuristics. For example, the dynamic programming algorithm from the previous section grows exponentially in space and time with k, the number of peaks in the spectrum. Hence, we cannot process spectra with too many peaks, as this would lead to out-of-memory errors or simply require more time than we are willing to invest. But we can choose an arbitrary k, such as $k = 10$ or $k = 15$, and run the dynamic programming algorithm restricted to the ten or 15 most intense or highest-scoring (see Sec. 9.6) peaks in the tandem mass spectrum. For the ILP approach, it is noteworthy that ILP solvers return (potentially suboptimal) solutions as soon as these are found; in many cases, an optimal solution is found much earlier than a proof that it is indeed an optimal solution.

9.6 Edge weights in the fragmentation graph

As noted, we have to weight our graph so that we indeed compute fragmentation trees and not fragmentation bushes. The first part of the score is, as usual, dealing with peak intensities and mass deviations: Each color corresponds to a peak in the query spectrum, and we can use the intensity of the peak to score all nodes with this color (Sec. 4.9). Furthermore, each node is the molecular formula of a hypothetical fragment, and we can use the mass deviation between the hypothetical fragment and the assigned peak to modify the score of the node (Sec. 4.7).

Both of the above are node scores, whereas our formulation of the Maximum Colorful Subtree problem assumes that only edges are weighted. But this is not an issue: In the fragmentation graph, we simply shift the node score to the weight of *all* incoming edges. In every subtree of the fragmentation graph, exactly one incoming edge is selected for each node that is part of the subtree. The only exception to this rule is the root of the fragmentation graph, but we demand that the root is part of all subtrees: Consequently, this score modification applies to all subtrees of the current fragmentation graph. After we have computed the maximum colorful subtree — in a graph with edge weights only — we add the score of the root to the subtree score. This is necessary if we want to compare subtrees from different fragmentation graphs, see Sec. 9.7.

But scoring nodes (peaks) exclusively, *any* subtree that encompasses the same nodes as an optimal subtree will be co-optimal. This includes fragmentation bushes from Sec. 9.1 — recall that the fragmentation graph is transitive. For this reason, we have to add *prior knowledge* to our computations. A word of warning: See Chapter 12 on the many things that can go wrong if you include prior knowledge in your computations. To lower the possibility of overfitting, we *must not* choose any parameters of the fragmentation tree computation so that, say, molecular formula identification rates are maximized.

What prior information can we use to weight the edges of the fragmentation graph? It is well known that certain losses are observed frequently in tandem mass spectrometry: These *common* losses include H_2O, CH_3, and CO. Similarly, many losses are included in the fragmentation graph which are thought to be implausible: This includes all radical losses but a few "common radical losses" such as $H^{\cdot}$, $O^{\cdot}$, or

·OH, and nitrogen-only or carbon-only losses. We can ask an expert to compile lists of common and implausible losses; an even better approach would be to learn such lists from the data. (Learning common losses from data is indeed possible, but to learn implausible losses is not.) Another observation we can make is that certain loss masses appear more often than others, beyond the common losses mentioned above: A log-normal distribution with median at 55.84 Da shows excellent agreement with observed loss masses. Finding the distribution and its parameters is highly non-trivial, as we have to include the prior knowledge into the fragmentation tree computation, which we then use to determine the parameters of the distribution. This is best carried out using an iterative approach similar to expectation maximization, where we compute fragmentation trees for a given reference dataset, then determine parameters from these fragmentation trees, and then iterate until (hopefully) the process converges. At the end, we can check whether our assumption — say, loss masses follow a log-normal distribution — is supported by the data. I will leave out the tedious technical details.

9.7 Finding the precursor molecular formula

Throughout all of this chapter, we have assumed that we know the molecular formula of the precursor ion, when in reality, that is what we want to determine. But considering the rest of this textbook, it is not hard to come up with a solution: We iterate over all possible molecular formulas that explain the precursor peak, or the corresponding peak from MS1. For each candidate molecular formula, we generate the fragmentation graph and solve the Maximum Colorful Subtree problem. We use the score (weight) of the best tree as the score of our candidate molecular formula, and then sort candidates by this score.

An alternative route — which ends up with exactly the same solution — is to integrate all molecular formula candidates for the precursor ion in the fragmentation graph simultaneously. All of these nodes have the same color; we add a superroot with a new color which is connected to each of the precursor ion nodes. The optimum tree in this graph is also the optimum tree we obtain by iterating over the molecular formula candidates. But this approach has a number of disadvantages compared to the iterative computation: First, we only get to know the top-scoring molecular formula candidate, not a ranking of all molecular formula candidates. This is undesirable since we often cannot expect that the true answer is scored highest, but rather have to consider the top 2 or top 10. Second, remember that the Maximum Colorful Subtree problem is NP-hard. Adding only a few nodes and colors can increase running times dramatically, such as trillion-fold or worse; the time we save by not iterating over the candidates is tiny in comparison. You might think that adding only one color (for the superroot) and a few nodes (for the molecular formula candidates of the precursor ion) does not make much of a difference, but the reality is much worse: If we consider molecular formula candidates one by one, we have to consider — for each candidate — only those molecular formulas for the fragments which are subformulas of the candidate formula. This will not only substantially reduce the number of nodes; in addition, we will usually find that some peaks cannot be explained as subformulas, reducing the number of colors. In total, combining all molecular formula candidates in one fragmentation graph is rather of theoretical algorithmic interest, as it will result in instances of the Maximum Colorful Subtree problem which are at the limit of what is feasible.

9.8 Historical notes and further reading

Fragmentation trees were introduced by Böcker and Rasche [36] in 2008. NP-hardness of the Maximum Colorful Subtree problem was shown independently by Fellows *et al.* [108] and Böcker and Rasche [36]: In the first paper, the more general Graph Motif problem is considered, which is then restricted to colorful motifs and trees as input graphs, resulting in the unweighted Maximum Colorful Subtree problem. In fact, the problem is also hard to approximate [114, 286].

We have been slightly sloppy when defining the Maximum Colorful Subtree problem: First, does the root of the graph have to be a node of the subtree? This is the case when we compute fragmentation

trees, but what about the general case? This point does not make a difference for the complexity of the problem, as we can easily transform the two problems into each other with polynomial (linear) overhead. Second, be reminded that we talk about subtrees when we are only interested in arboreal (that is, tree-like) subtrees. Finally, it is noteworthy that neither the transitivity of the fragmentation graph, nor the fact that its coloring is order-preserving [114], have been used in the design of the exact algorithms.

For an introduction to parameterized algorithms, I refer the reader to [85, 253]. Using subsets of colors as part of the dynamic programming recurrence, has been used frequently in algorithmics: See for example Dreyfus and Wagner [87] who, back in 1972, used subsets of terminal nodes in a dynamic programming recurrence to compute a shortest Steiner tree. As shown in Sec. 9.4.1, the dynamic programming algorithm has running time $O(2^k \cdot m + 3^k \cdot kn)$. For integer edge weights of absolute value at most M, running time can be improved to $O(2^k \cdot M \cdot p(n,k))$ where $p(n,k)$ is a polynomial in n and k, using the Möbius transform and the inversion technique of Björklund *et al.* [24]; but this is rather of theoretical interest, in particular since we can no longer use the space- and time-saving algorithm engineering tricks.

Numerous books have been written about integer linear programming, see for example Schrijver [314]. The applications of (integer) linear programming are unlimited and include spacecraft trajectory planning [288] and smart homes [332]. To get a feeling for the importance of integer linear programming, take a look at the price lists of commercial solvers: A solver may easily cost US$ 50k per year. If you want to know more about the difficulties of beating an ILP formulation using a "custom-made" algorithm, see for example Böcker *et al.* [27] for the Cluster Editing problem and Rauf *et al.* [286] for the Maximum Colorful Subtree problem.

My presentation of heuristics for the Maximum Colorful Subtree problem follows Dührkop, Lataretu, White and Böcker [91], see there for further details. It turns out that the Critical Path heuristic performs on par with the exact methods when it comes to molecular formula identification, and that two variants of this heuristic perform even better than exact methods. Somewhat unexpectedly, the two spanning tree heuristics have very different performance, both with regard to running time and identification performance. Yet even for the best heuristic, more than 20 % of the computed trees differ from the optimal tree, often significantly so; this argues against using heuristic trees for downstream analysis.

Why do we care about the structure of the fragmentation tree when evaluating heuristics for the problem? First, Rasche *et al.* [285] showed that the structure of fragmentation trees — as computed by an exact method — agrees well with what a human expert would assign. But second, fragmentation trees contain valuable information about the structure of a compound which goes beyond that available in the fragmentation spectra! This was first shown by Rasche *et al.* [284], who introduced fragmentation tree alignments (yet another NP-hard problem) for the pairwise comparison of compounds. Today, the pairwise comparison of fragmentation trees forms the basis of CSI:FingerID [93, 323]; replacing fragmentation *spectra* comparison in FingerID [149] with fragmentation *tree* comparison in CSI:FingerID is the driving force behind the huge performance leap between these tools.

For scoring fragmentation trees, Böcker and Rasche [36] initially proposed an *ad hoc* scoring where the list of common losses was indeed provided by a mass spectrometry expert. Böcker and Dührkop [28] replaced the *ad hoc* scores by a statistical model where all parameters were instead learned from data. To avoid overfitting, the statistical model uses an intentionally small number of parameters. It is noteworthy that the *algorithms* for computing fragmentation trees stay the same; only the edge weights in the fragmentation graph change. Although parameters were not chosen in a way to maximize the molecular formula identification rate for the precursor ion, this rate nevertheless almost doubled with the new edge weights.

Scheubert *et al.* [312] modified the approach of calculating fragmentation trees, so that MS^n data can be taken into account. Interestingly, whereas it appears to be a much simpler task to reconstruct such trees from multiple MS data, the computational questions that arise in this context are even harder than those presented here. In practice, this is not much of an issue, after applying a decent amount of algorithm engineering [310].

Tandem mass spectra of a small molecule can be measured at different collision energies: Higher energies lead to smaller fragments, as more chemical bonds break. To increase the amount of information

available to us, we may demand that several mass spectra, measured at different fragmentation energies, are given to us. This is different from peptide fragmentation, where spectra are always measured using one "fixed" collision energy that breaks the peptide backbone. (Notably, modified peptides such as glycopeptides are sometimes measured at a second collision energy, to also gain information about the modification.) We can merge multiple spectra from the same metabolite into one, merging peaks from different spectra with masses "sufficiently close." Alternatively, one tandem mass spectrum can be measured in "ramp mode", where the collision energy is varied while measuring the spectrum.

9.9 Exercises

9.1 How many subformulas exist for molecular formula $C_6H_{12}O_6$? Find a general formula.

9.2 Why is it inevitable that we demand that the subtree of the fragmentation graph has to be colorful, when defining the Maximum Colorful Subtree problem? Describe what happens if we drop this requirement.

9.3 Why do some edges of the fragmentation graph usually have negative weight, when defining the Maximum Colorful Subtree problem?

9.4 Show that a directed graph T is a tree (formally, an arborescence) with root r if and only if every node v of T can be reached from r, and every node but the root has in-degree one, whereas the root has in-degree zero.

9.5 Show that a directed graph T is a tree (an arborescence) if and only if it is acyclic, and every node but the root has in-degree one, whereas the root has in-degree zero.

9.6 Assume that a company wants to produce two products, A and B. You need three machines to produce these products: Product A requires 7 minutes of processing time on machine 1, 4 minutes on machine 2 and 3 minutes on machine 3; product B requires 4 minutes on machine 1, 9 minutes on machine 2 and 3 minutes on machine 3. We can sell product A for 9 \$ and product B for 13 \$. Maximize the income you can make in one hour using linear programming.

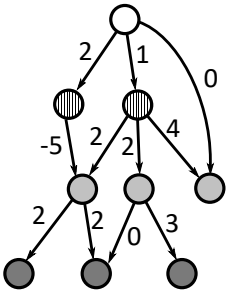
9.7 Prove that the *Kruskal-style* heuristic always returns a subtree.

9.8 Give a simple example where the *Kruskal-style* heuristic and the *Prim-style* heuristic return different trees.

9.9 Show the correctness of recurrence (9.6).

9.10 Show the correctness of recurrence (9.7); give an example why the recurrence is no longer correct if the coloring of G is not order-preserving.

9.11 For the fragmentation graph shown below, compute the heuristic solutions for the *Kruskal-style* and *Prim-style* heuristic, with and without *Remove Dangling Subtrees* postprocessing. Assume that all transitive edges (for example, from the root to any other node) which are not shown, are also present in the input graph but have weight -10.



9.12 For the fragmentation graph from Exercise 9.11, compute the *Critical Path* heuristic. Compare all heuristic solutions to the exact solution of the MAXIMUM COLORFUL SUBTREE problem.

★ 9.13 Prove that finding a maximum subtree in a DAG that may contain negative edges, is an NP-hard problem.

10 Searching metabolite structure databases

> *"Nature makes penicillin; I just found it."* (attributed to Alexander Fleming)

> *"You want to know how two chemicals interact, do you ask them? No. They're gonna lie through their lying little chemical teeth. Throw them in a beaker and apply heat."* (Dr. Gregory House)

Fragmentation tree computation, as introduced in the previous chapter, can be an important first step in the analysis of fragmentation data from small molecules. But ultimately, the thing that we want to know is the molecule's *structure*. We will see in Sec. 10.4 that *de novo* structure elucidation of small molecules is impossible, in contrast to peptides and glycans (Chapters 2 and 11). This does not mean that computational mass spectrometry cannot *assist* in doing so, but a lot of prior background knowledge and potentially additional experimental data has to be added. For this reason, we will concentrate on searching in molecular structure databases (Sec. 10.3). Note that searching in spectral libraries — *the* standard way of identifying small molecules via mass spectrometry, sometimes referred to as "dereplication" — has been covered in Chapter 4.

10.1 More facts about metabolites

I assume that you, the reader, are familiar with proteins and peptides — this is taught, say, to undergraduate students from bioinformatics. But I also assume that you are less familiar with "metabolites"; this is why I want to collect a few facts about them. If you already know what I am talking about, skip this section.

Metabolites can be subdivided into two major classes. A *primary* metabolite is directly involved in growth, development, and reproduction of a cell or organism: For example, adenosine-5'-triphosphate (ATP) is the energy currency of the cell. A secondary metabolite is not directly involved in those processes. Examples include antibiotics and pigments; a secondary metabolite of particular importance to science in general, is shown in Fig. 10.1.

A major challenge is that most of the secondary metabolites in any given higher eukaryote are largely unknown: Current estimates are in the range of up to 20 000 metabolites for any given species. In particular, plants, filamentous fungi, and marine bacteria synthesize enormous numbers of secondary metabolites. Unlike for proteins, genome sequencing usually does not allow us to deduce the structure of the metabolites.

Another challenge that we have to face, is that the molecular structure of metabolites is not restricted: Unlike for biopolymers which are made from smaller monomer building blocks in some ordered fashion (strings for proteins, trees for glycans) the molecular structure of metabolites is not restricted. In spite of the small size of metabolites, this results in a huge variety and complexity of such molecules.

For the analysis via mass spectrometry, the important point is not that metabolites are intermediates or products of some metabolism; the important point is that they are *small molecules*: These are molecules with mass below about 1000 Da that, by definition, are not biopolymers. No rule without exception: Small biopolymers such as peptides with only two amino acids (for example, carnosine) or disaccharides (for example, sucrose) are sometimes also considered metabolites. It is understood that there exist small molecules that are not metabolites: To name two examples, consider drugs and pesticides, which are often synthetic. To include these small molecules, we will speak about "biomolecules", that is, molecules that are products of nature, or synthetic products with potential bioactivity. For mass spectrometry, there is

usually no fundamental difference analyzing metabolites or other biomolecules. So, when we speak about "metabolites", you can often replace this by "small biomolecules".

Much like for proteins, mass spectrometry is also a key technology for the identification of small molecules. Various analytical setups have been developed, most notably gas chromatography MS (GC-MS, also GC/MS) and liquid chromatography MS (LC-MS, see Sec. 1.6.2). GC-MS is almost exclusively coupled with Electron Ionization (EI), which both ionizes and fragments molecules. GC-MS spectra are usually interpreted via database search in spectral libraries of references. To generate an entry in a spectral library, someone has bought the (mostly pure) molecule from a vendor, then measured it on their GC-MS instrument, recording its retention time and fragmentation spectrum. In this context, the molecule is referred to as a "standard" or "reference compound". Fragmentation spectra of reference compounds have been collected over many decades. Spectral library search is comparatively easy as the fragmentation is highly reproducible across instruments, vendors, and time. Large spectral libraries of measured GC-MS reference spectra are available, such as the commercial library from the National Institute of Standards and Technology (NIST/EPA/NIH Mass Spectral Library). A major issue of searching GC-MS libraries is that the fragmentation is executed without precursor selection, so that the mass of the molecule is unknown and we have to search the complete library without filtering. On the other extreme, EI fragmentation spectra are often interpreted by hand, and numerous books have been written that explain how to do this. This is possible as EI fragmentation is comparatively easy to understand, reproduce and simulate.

GC-MS requires metabolites to be thermally stable. Unfortunately, this is not the case for several biologically important compound classes: Just imagine what happens to sugar-containing metabolites when you heat them. For the analysis via GC, molecules have to be derivatized; this is done to improve the volatility of compounds, to reduce polar substances in polarity or to increase sensitivity. For example, the hydroxy groups of sugars are silylated to make them available for the analysis via GC-MS. Alternatively, these molecules can be analyzed using LC-MS. Recall the experimental setup from Sec. 1.6.2: First, molecules are separated by liquid chromatography, and MS1 spectra are recorded; the precursor ion of one molecule is mass selected, and fragmented by collision-induced dissociation; and masses of resulting fragments are recorded in a tandem mass spectrum. Compared to GC-MS, this has the additional advantage that we can fragment one molecule at a time, whereas fragmentation by EI is applied to all molecules that leave the GC column simultaneously. The manual analysis of small-molecule tandem MS data is highly non-trivial and time-consuming, as the fragmentation of small molecules under varying fragmentation energies is not completely understood. Tandem MS of small molecules is *highly reproducible* if you compare spectra measured on the same instrument. On different instruments and, in particular, when comparing tandem mass spectra from different instrument types such as Orbitrap and QTOF, reproducibility is much lower than for EI fragmentation. But even in this case, reproducibility is high enough to allow for spectral library searching.

As GC-MS is very important for analyzing small molecules, we will usually speak about the *fragmentation spectrum* of the molecule, which can be its tandem mass spectrum, its electron ionization fragmentation spectrum, or anything else you can think of. Clearly, fragmentation processes are very different and sometimes require massive modifications of an approach; but many general ideas can be easily transferred.

10.2 Representing molecular structures as graphs

There exist at least two ways to encode the structure of a molecule as a graph, both being absolutely straightforward. The *molecular graph* $G = (V, E; c)$ of a given molecular structure represents each atom (including hydrogen atoms, which are frequently omitted from molecular structure drawings) by a node. Each node is labeled by the corresponding element through the function $c : V \to C$, where C is the set of all chemical elements. The graph is undirected but not simple, meaning that two nodes can be connected by more than one edge. For an edge between u and v, we will use the notation $uv \in E$. For every covalent bond in the molecular structure, we insert edges into the graph according to the bond order: A single bond

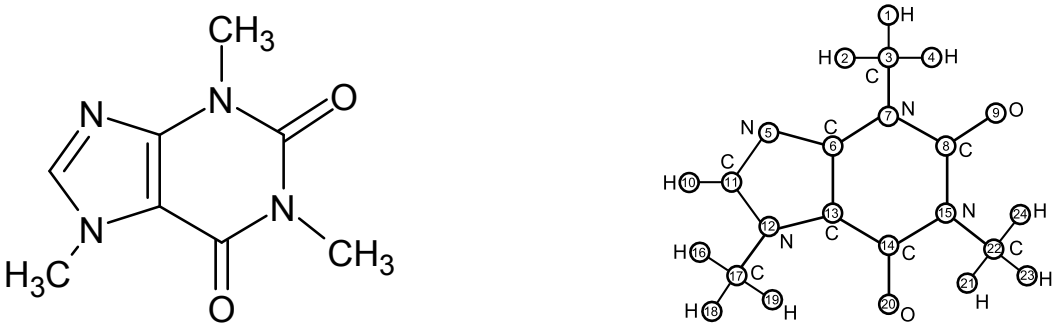


Figure 10.1: Left: Structural formula of caffeine, molecular formula $C_8H_{10}N_4O_2$, monoisotopic mass 194.080376 Da. Right: Corresponding molecular graph with colors from $\{C, H, N, O\}$, but without bond orders. Note that some hydrogen atoms are omitted in the structural formula.

is represented by one edge, a double bond by two edges, and a triple bond by three edges. Clearly, doing so we cannot represent, say, aromatic bonds in the graph; aromatic rings have to be resolved into single and double edges.

The *bond-labeled molecular graph* $G = (V, E; c, b)$ again represents all atoms as nodes and labels them accordingly. But this is a simple undirected graph, meaning that two nodes can be connected by at most one edge. In addition, each edge e is mapped to some bond order $b(e)$, which besides single, double and triple bonds may also be an aromatic edge, or anything else that comes to mind. See Fig. 10.1 for an example; bond orders are omitted from the graph.

A *path* $p = u_0u_1 \dots u_l$ in an undirected graph G is a list of nodes $u_0, u_1, \dots, u_l \in V$ such that $u_{i-1}u_i \in E$ is an edge, for all $i = 1, \dots, l$. We say that p is a path *from* u_0 *to* u_l of *length* $|p| := l$. A graph G is connected if, for any pair of nodes u, v, there exists a path from u to v. Clearly, there can be more than one such path. Recall that only molecules held together by covalent bonds will be measured jointly in the MS instrument. In other words: If the molecular graph is not connected, it will not be measured in MS as one.

To deal with fragmentation mass spectra, we consider subgraphs of the molecular graph. Any graph $H = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$ is a *subgraph* of $G = (V, E)$. Subgraph H is an *induced* subgraph (more precisely, a node-induced subgraph) if $E' = E \cap \binom{V'}{2}$, where $\binom{X}{2} = \{\{x, y\} : x, y \in X, x \neq y\}$ is the set of subsets of cardinality two. Colloquially speaking, an induced subgraph contains all the edges from the original graph connecting nodes from V'. A *cut* $F \subseteq E$ is a subset of edges such that the graph $(V, E - F)$ is not connected. For an induced subgraph (V', E'), the corresponding cut is

$$F := \{uv : u \in V', v \in V - V'\}.$$

Recall that the weight of a fragment is determined by its nodes (atoms), not its edges (bonds). On the other hand, breaking a bond requires energy. For this reason, we will often restrict ourselves to induced subgraphs.

Going from strings (peptides) or trees (glycans) to arbitrary graphs comes with numerous computational problems. For example, even checking whether two representations of a molecular graph are actually "the same", is a highly non-trivial problem: This is the graph isomorphism problem, and it is famous in complexity theory for being a candidate for the complexity class "NP-intermediate". The subgraph isomorphism problem is simply NP-complete.

Molecular graphs correspond to the "constitution" of a molecule, but do not account for its 3-dimensional structure. For any given molecular graph, there can exist numerous *stereoisomers* which differ solely in the 3-dimensional orientations of their atoms in space. Thalidomide is a particularly infamous example demonstrating the importance of stereochemistry; D-serine, an enantiomer of the common L-serine, serves as a neuromodulator. It is widely believed that differentiating between stereoisomers is *currently* and *in general* beyond the power of mass spectrometry. (For certain hand-selected examples it may be possible today; and this may change in the future.) In this regard, molecular graphs, representing

the "2-dimensional structure" of the molecule, are an adequate representation. I have set "2-dimensional structure" into quotation marks because certain molecular graphs require the third dimension to be "reasonably" drawn, see tetrodotoxin (fugu poison) as an example. Note that the molecular graph of tetrodotoxin is nevertheless *planar*, as it contains neither K_5 nor $K_{3,3}$ as a minor. But to embed the tetrodotoxin molecular graph into the plane requires highly unequal edge lengths. In fact, even the molecular graph of buckminsterfullerene (the bucky ball) is planar.

Do not ask me for a definition of *aromaticity*; in short, it is complicated. It is enough for us to know that certain chordless cycles in the structure can be labeled "aromatic ring". Aromatic rings are usually made from five or six atoms. These rings do not break apart easily, which is obviously of interest for us when we try to interpret the fragmentation of a small molecule. Deciding whether a particular ring in a molecular structure is an aromatic ring, is a non-trivial problem [233, 245, 313]; the often-applied Hückel rule can, for example, only be applied if there is only one ring.

There exist many issues with representing a molecular structure as a molecular graph. For example, "tautomerism, mesomerism and alternate ionization states contribute to the number of possible valid, nonidentical representations of the same structure" [142]; sometimes, these representations exist in equilibrium [308]. These issues are somewhat cumbersome to deal with, see for example Hähnke *et al.* [142] on the standardization of molecular structures in PubChem. At a certain stage, the molecular graph representation may result in wrong conclusions being drawn about the molecule. (For molecular graphs with multiple edges between nodes, we cannot even represent aromatic bonds; resolving them as single and double bonds already results in ambiguity that we have to deal with later.) Nevertheless, molecular graphs are extremely helpful for many applications, and have the advantage of simplicity: If you want to avoid such issues, use quantum chemistry calculations. Citing George E. P. Box once more: "All models are wrong but some are useful."

10.3 Searching in metabolite structure databases

In the following, our goal is to structurally elucidate a small molecule (e.g., a metabolite), given its tandem mass spectrum. Unfortunately, to do so *de novo* is not possible, see Sec. 10.4. (It may be possible for a few hand-selected cases, but definitely not *in general*.) This is why searching in spectral libraries has been the standard way of identifying metabolites for decades.

In the following, we want to replace searching in spectral libraries by searching in molecular structure databases — but why? Spectral libraries are (and will be) tiny, in comparison to the huge molecular structure databases. PubChem already contains more than 100 million compounds and is steadily growing. In comparison, spectral libraries usually contain thousands to tens of thousands of (singular) compounds. The actual numbers are of no importance here; it is enough to see that structure databases are orders of magnitude larger. There is no reason to believe that this gap will get smaller in the future: All biologically relevant standards that you can buy for little money, have already been included in spectral libraries. In fact, different spectral libraries today show an overlap which may be surprising, given that so many biomolecules are still absent from all libraries. In the near future, people will search in structure databases that contain *hypothetical* metabolites; when you read this, it may already be an accepted standard procedure.

Searching in a structure database proceeds in the usual way: First, we filter all molecules from the database that have the correct precursor mass or molecular formula; these are our candidates. Second, we then rank all candidates using the measured spectrum; this is usually done by computing some score. Currently, there exist four paradigms to rank the candidates:

- **Structure to mass spectrum.** We transform a molecular structure (for example, a molecular graph) into a fragmentation spectrum. The advantage here is that we can do the transformation as preprocessing; the actual molecular structure search boils down to a spectral library search, where spectra in the library are all simulated. This is closest to what we know from peptide mass spectrometry, where — in its simplest incarnation — simulating a fragmentation spectrum is trivial.

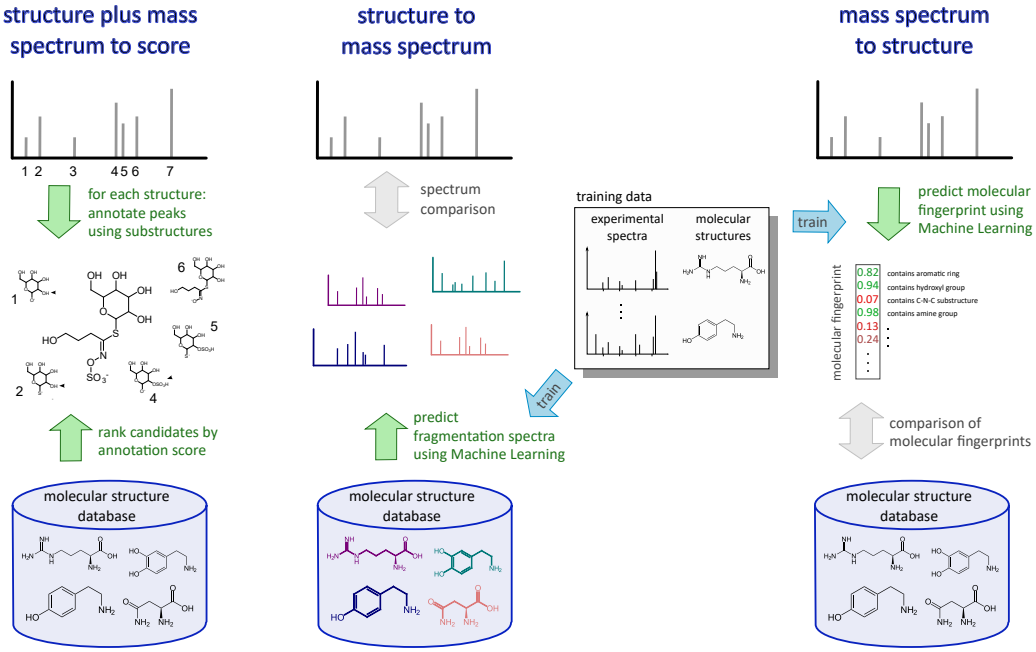


Figure 10.2: Three paradigms for searching in small molecule structure databases.

Unfortunately, it is highly non-trivial for small molecules. Current approaches are either based on quantum chemistry calculations (EI fragmentation) or machine learning (tandem MS and EI fragmentation); for lipid classes such as sphingolipids, rule-based approaches are still state of the art. To give you an idea of how involved this can get: The machine learning approach CFM combines expectation maximization, neural nets and a hint of Markov chains [1, 2]. It must be understood that the simulated spectrum of a compound is usually rather distinct from its experimental counterpart; but the primary task here is to rank the correct candidate at a top position (see Sec. 4.11), not to simulate a nice-looking spectrum.

- **Structure plus mass spectrum to score.** Here, we try to explain the peaks in the measured fragmentation spectrum as induced subgraphs of the candidate molecular graph. This is also known as *combinatorial fragmentation*, as it does not simulate the actual fragmentation of the molecule — we would not need a measured spectrum to simulate a spectrum — but instead, simply tests which of the detected peaks can be "explained away". Obviously, our score can take into account mass deviations (Sec. 4.7) and peak intensities; but since there is no simulated spectrum, we have to resort to scoring against barcode spectra (Sec. 4.9). This paradigm is conceptually the simplest of the first three, but comes with the disadvantage that both the structure candidate and the experimental query spectrum have to be known to compute a score. We will come back to this in Sec. 10.5.

- **Mass spectrum to structure.** We transform the query fragmentation spectrum into information about the structure of the query molecular structure (usually, a molecular fingerprint). The advantage of this paradigm is that this transformation does not require a database. The molecular structure search boils down to comparing the structure information (say, the molecular fingerprint) to structures in a molecular structure database. But the method is not limited to search in a molecular structure database, and we can also use the predicted structure information to derive further information about the query molecular structure. Current approaches use machine learning to derive the structure information, such as kernel-based support vector machines (CSI:FingerID) or deep neural networks.

- **Guilt by association.** These are meta-methods that try to extend the annotations of some identification method (spectral library search, sometimes even searching exact match) to other queries

which were initially not identified. To do so, they require that some molecular network is known. In *molecular networking*, queries are connected if their fragmentation spectra are similar [361, 363], assuming that similar spectra stem from structurally similar compounds; annotations can then be "propagated" along the edges of the network [68]. Alternatively, compounds can be connected via known biotransformations, for example using a metabolic reaction network [324].

Claude helped!

See Fig. 10.2 for the first three paradigms. These three paradigms are referred to as "*in silico* fragmentation", although this name only fits the first paradigm; the other two paradigms do not (claim to) simulate small molecule fragmentation *in silico*. The "structure to mass spectrum" paradigm is closest to what biochemists are used to from spectral library search; but as detailed in Sec. 4.11, this does not mean that it is the best way to rank candidates.

Finally, what databases do we search in? There exist large databases such as PubChem (https://pubchem.ncbi.nlm.nih.gov/) or ChemSpider (http://www.chemspider.com/) which contain millions of compounds. Searching there has the disadvantage that only a small fraction of the entries are biomolecules. Also, searching in a large database is considerably harder than searching in a small one, as bogus hits (high-scoring wrong candidates which, just by chance, have a better score than the correct candidate) will increase, see Chapter 6. On the other hand, there are specialized databases such as HMDB (http://www.hmdb.ca/) or ChEBI (https://www.ebi.ac.uk/chebi/) which concentrate on a subset of biomolecules. Unfortunately, there is no biomolecule database containing all of them, so one has to aggregate the different databases "by hand". *In practice,* a reasonable strategy is to search first in a biomolecule database; only queries where we do not find a reasonable hit, are then searched in PubChem or ChemSpider. See Chapter 12 for a justification of this approach. *For evaluations,* it makes most sense to search PubChem or ChemSpider, as this is more challenging; this is what you will (hopefully) see in publications from computational mass spectrometry on the topic.

10.4 *De novo* structural elucidation of small molecules

What about metabolite "*de novo* sequencing"? For peptides (Chapter 2) and glycans (Chapter 11) it is possible, at least in principle, to determine the structure without having to rely on a database to search in. Is something similar possible for small molecules?

Unfortunately, this is not possible. This is so for a number of reasons, some of which have been discussed previously:

- Given the structure of a metabolite, it is highly involved to simulate its spectrum.
- Given one or more peaks in the fragmentation spectrum, it is highly involved to deduce information about the metabolite's structure.
- Fragmentation depends on the energy we are applying; and since some metabolites show informative fragmentation for certain fragmentation energies only, we have to solve the above problems for any given energy.
- Many metabolites do not show informative fragmentation at any energy; a fragmentation spectrum with fewer than ten peaks is rather common.
- Metabolites with different structures can have practically indistinguishable fragmentation spectra. We observe this phenomenon already today, although our spectral libraries from chemical standards are puny, in comparison to all biomolecules that exist, or molecular structures that might exist.
- Genomic data tells us little about the structure of metabolites.
- Whereas the structure of peptides (strings) and glycans (trees) is highly restricted, very few restrictions exist for metabolites (graphs).

- Consequently, whereas the number of peptides and glycans (see Chapter 11) may appear intimidating, numbers are even worse for metabolites.
- We do not even know how many metabolite structures exist for a given molecular formula, up to a given mass, or any related question. At present, the only way to compute this number, is to actually generate (enumerate) each structure.
- Testing if two molecular graphs are actually the same molecular structure, is already a highly nontrivial problem. Testing for substructures is even worse.

Recall that you may exchange "metabolite" for "small biomolecules". The above does not mean that computer programs cannot *assist* in the structural elucidation of a metabolite; but a full structural elucidation by mass spectrometry is, by all accounts, impossible. (I am happy to take bets here.) The above statements hold *in general*: You will find certain metabolites or metabolite classes where this is not the case, but generalizability dictates that this is not enough.

The best shot at the problem is to combine searching in structure databases with structure generators such as MOLGEN, the Open Molecule Generator, or Surge. Structure generators enumerate, for a given molecular formula or mass, all molecular structures that are chemically sound. (Molecular structure enumeration is in itself a scientifically extremely challenging task, and many master's and PhD theses [139, 366] — often under the supervision of Prof. Reinhard Laue and Prof. Adalbert Kerber at the University of Bayreuth — and books have been written about this topic [186].) This allows us to overcome the boundaries of database searching: Generate all molecular structures corresponding to the precursor mass or molecular formula, and use the output of the structure generator as our database, see Sec. 2.8. But unfortunately, there is little hope that this approach will ever work, at least in this simple setup: The problem is not the structure generators, which can enumerate millions of structures in a matter of seconds; the problem is the size of the search space. For example, MOLGEN enumerates *all molecular structures* with molecular formula $C_8H_6N_2O$, in a matter of minutes [183]. Unfortunately, it turns out that there are *more than 100 million such structures*. Now, ranking these structures to find the one that is correct, is an extremely hard task: Ranking the candidates will take considerable time; but even worse, ranking the correct answer at a top position is extremely challenging. Remember that PubChem also contains more than 100 million structures; but using a molecular structure generator, all structures have the same molecular formula, and we *cannot filter any candidates* based on the molecular formula! Finally, the molecular formula $C_8H_6N_2O$ with mass 146.048013 does not correspond to a particularly large biomolecule; things get even worse for larger molecules, as we will see below.

So, how many molecular structures exist for a given (nominal) mass? As noted, we do not have a clue, except for certain special cases. An obvious but highly misleading approach is to look into molecular structure databases such as PubChem; but what we observe there is that after a certain mass, the number of structures *decreases*. Another possibility is to consider only a particular class of small molecules: The combinatorially simplest class are alkanes with molecular formula C_nH_{2n+2}. From a combinatorial standpoint, alkanes correspond to trees with maximum degree 4; this combinatorial standpoint ignores whether the molecular structures can exist in 3D space, but is "as good as we can get". The number $t[n]$ of alkanes C_nH_{2n+2} can be computed exactly using a recurrence, but also approximated as

$$t[n] \sim 0.6563186958 \cdot n^{-5/2} \cdot 2.815460033^n, \tag{10.1}$$

see Sec. 11.7. For example, for molecular formula $C_{71}H_{144}$ with nominal mass 996 there exist $1.28 \cdot 10^{27}$ alkanes. Finally, we can indeed enumerate all structures to count them. Clearly, we have to restrict our alphabet of elements to limit the combinatorial explosion. For example, using the alphabet of elements CHNO and nominal mass 150 there exist 615 977 591 molecular structures. Although we do not have any approximation for the number of molecular structures with nominal mass m over CHNO, we can roughly extrapolate the number as

$$2.0668257909 \cdot 10^{-4} \cdot 1.2106212044^m. \tag{10.2}$$

This number *does not come* with any guarantee of convergence (Sec. 15.7), but is just a crude regression of the available numbers. Remember the proverbial "Milchmädchenrechnung" from the footnote on page 123: We know it is wrong but it can nevertheless be very instructive. Assuming that (10.2) is accurate, the number of molecular structures increases 1000-fold every 36 Da; in particular, we have $2.11 \cdot 10^{79}$ molecular structures for nominal mass 1000 over the alphabet CHNO. This is roughly the estimated number of atoms in the observable universe. The number does not change much if we consider all masses *up to* 1000 Da instead of *exactly* 1000 Da, see Exercise 10.3. If we include phosphorus, sulfur and halogens in this estimate, things will get worse, but it is hard to estimate how much worse — the number is probably somewhere between 10^{85} and 10^{110} molecular structures for mass up to 1000 Da, potentially closer to the lower number. (One might expect that adding these elements would result in an even bigger combinatorial explosion, but this appears not to be the case: For the alphabet of elements CHNOSiPSFClBrI and nominal mass 150 there exist 1 052 647 246 molecular structures, only 71 % more than for the alphabet CHNO.) For comparison, there exist $1.10 \cdot 10^{26}$ peptides of length up to 20; you will have to go up to peptide length 61 to be a match for the number of metabolite structures.

10.5 Matching masses to molecular substructures

Let us assume for a moment that we have both the experimental tandem MS data, *and* the correct molecular structure of the molecule: All we have to do, is to match what we see experimentally with the known structure. You might wonder whether this is a little bit too much information at the same time. As CID fragmentation of metabolites is still largely not understood, it is actually a rather sensible question. But in practice, this "oracle" will be used as a subroutine of the combinatorial fragmentation approach for searching in molecular structure databases.

Unfortunately, many negative (hardness) results are waiting along the path to this problem. For this reason, let us focus on the potentially most basic problem: Given a molecular graph, is there a connected subgraph of mass m?

To make things somewhat simpler for us, let us assume that we have successfully transformed the mass of the fragment into its molecular formula. Then, the question is: Is there a substructure of the molecule with the given molecular formula? We now formalize this problem, see Sec. 10.2: We are given a node-colored, undirected graph $G = (V, E)$. A molecular formula corresponds to a *multiset* of colors: A multiset is a set where, for each element in the set, we also record the number of times it appears. Now, our question is: Is there a connected induced subgraph G' of the given graph, such that the multiset of colors of this subgraph equals a given input multiset of colors?

In theoretical computer science, this problem is known as the Graph Motif problem. Unfortunately, a series of papers has proven the hardness of the problem with respect to different algorithmic flavors. We just note that the problem is NP-hard even for bipartite graphs of maximum node degree four and only two colors. On the positive side, several parameterized algorithms were constructed. For the application that we have in mind, the Graph Motif problem is an oversimplification, as we ignore the cost of cutting out a fragment at a particular position; incorporating edge weights and solving the optimization problem leads to increased running time.

But somewhat counterintuitively, the problem that we address here, is *simpler* than the Graph Motif problem: As the fragmentation energy is limited, the molecule cannot break at too many positions simultaneously. So, we assume that we are given an upper bound b, such that at most b edge removals must suffice to disconnect the subgraph G' from the remainder of G. In practice, we may assume that this b is rather small, such as $b = 3$. Assume that we have some objective function $w : E \to \mathbb{R}$ that we want to minimize; w may correspond to the energy required to break some bond. Now, we ask: What is the induced subgraph G' of G that can be separated from the remainder of G via deleting a cut set $E' \subseteq E$ with $|E'| \leq b$, such that $w(E')$ is minimum? As we require $|E'| \leq b$, there may be no such graph.

We can try all edge sets with at most b edges, and see if they produce the desired multiset (molecular formula). As there are $O(|E|^b)$ such edge sets, an algorithm for that purpose will run in roughly that time.

But we want to do something slightly smarter: We use a branch-and-bound heuristic, aborting our search as soon as no optimal solution can be found in the future. Given a cut set $E' \subseteq E$ with $|E'| \leq b$, its deletion might separate G into a set of connected components. We use depth-first search to identify the connected components in G; simultaneously, colors are counted and costs for the partition are calculated. If the colors of a connected component correspond to the colors of the input multiset, and the costs are smaller than the costs of the best solution found so far, the connected component is stored. These costs w^* are then used as an upper bound for pruning.

Initially, edges are sorted in increasing order with respect to their weight. We use edge set iterators $i_1 < i_2 < \cdots < i_b$ pointing to the sorted set E. We iterate $i_1 = 1, \ldots, |E|$, and inside this loop iterate $i_2 = i_1 + 1, \ldots, |E|$, and so on, see Algorithm 8.1 on page 131. Let $w(i)$ be the cost of the edge that corresponds to iterator i. Assume that we have iterators $i_1, \ldots, i_{a-1}$ fixed, and that we want to iterate $i_a = i_{a-1} + 1, \ldots, |E|$ for $a \leq b$. Assume further that the weight of the partial solution for iterators $i_1, \ldots, i_{a-1}$ is known; we compute the weight of the partial solution for iterators $i_1, \ldots, i_a$ in constant time, adding $w(i_a)$, and pass this weight to the following iteration steps. We abort this loop if our current weight exceeds the current minimum weight w^*.

Another trick for speeding up this algorithm in applications, is to iterate over $b' = 1, \ldots, b$, and to examine only cut sets E' with $|E'| = b'$. Note that you may not stop if you found a solution for some $b' < b$, as cut sets with larger cardinality may have smaller weight. Another possibility is to provide not only b as the maximum number of edges we are allowed to break, but also some maximum weight w^* that we are allowed to use for breaking edges.

Analyzing the worst-case running time of our algorithm is quite simple, as none of the heuristic improvements discussed above, does anything good to the worst-case running time. Sorting edges costs $O(|E| \log |E|)$ time. Running time of the depth-first search is $O(|V|)$. The branch-and-bound algorithm iterates over $O(|E|^b)$ edge sets. This results in an overall running time of $O(|E| \log |E| + |V| \cdot |E|^b)$. It is easy to see that this algorithm also answers the question, whether there is a substructure of some given mass m.

10.6 Lipids

One large class of metabolites are lipids: These are biomolecules that are soluble in nonpolar solvents, and include fatty acids, waxes, sterols, fat-soluble vitamins, monoglycerides, diglycerides, triglycerides, and phospholipids. Unlike general metabolites, lipids are very restricted in their molecular structure: In particular, they often have long chains made from carbon and hydrogen, usually with only a few double bonds between the carbon atoms, which are attached to some backbone.

Unlike general metabolites, the fragmentation of lipids is relatively easy to predict. Usually, we cannot identify the exact positions of double bonds in the hydrocarbon chains; hence, what we want to identify is not the exact molecular structure, but rather a structure intermediate. Overall, it does not make sense to computationally analyze lipids in the same way that we do it with other small biomolecules: In this way, lipids are similar to peptides and glycans, where applying the generalist methods from this chapter would also be “overkill”.

LipidBlast and CFM-ID version 3 make use of the known fragmentation patterns of lipids, to predict lipid spectra via rules similar to what we do for peptides (abcxyz ions). The Lipid Data Analyzer (LDA) avoids simulating lipid mass spectra, and instead follows the “mass spectrum to structure” paradigm, transforming the mass spectrum into structure information about the lipid based on expert-curated decision rules. In evaluation, LDA substantially outperforms approaches based on simulating lipid mass spectra.

In short: Lipids are special; do not make things complicated by treating them as if they were general metabolites.

10.7 DENDRAL

The first rule-based approaches for predicting fragmentation patterns, as well as explaining experimental mass spectra with the help of a molecular structure, were developed as part of the DENDRAL project that started back in 1965 [206, 207]: This led to a series of papers, dealing with the interpretation of mass spectrometry data, and the identification of metabolites. Meta-DENDRAL was used to derive new "rules of thumb" for the analysis of mass spectrometry data, published in a series of papers in the *Journal of the American Chemical Society*. Gray *et al.* [133] describe a computer program that predicts mass spectra of candidate structures generated by CONGEN and ranks the candidates, which corresponds to Sec. 10.5. Lavanchy *et al.* [203] use this method to interpret mass spectra of marine sterols. Gray *et al.* [132] do a similar analysis using the related GENOA program. See Mun and Mclafferty [246] and Smith *et al.* [330] for reviews of different aspects of the DENDRAL project at that time. Other parts of the project dealt with the automated analysis of Nuclear Magnetic Resonance (NMR) data [134]. A full coverage of all the techniques developed as part of the DENDRAL project, is far beyond the scope of this textbook; see Chapter 7 of the PhD thesis of November [258] for the early years of the project. Some of the techniques used in DENDRAL would today go under the name "combinatorics", "algorithmics" or "graph theory": For example, computing molecular formulas from masses was achieved by a brute-force search with some simple pruning strategies, see Chapters 3 and 8. But at the time, the label "Artificial Intelligence" probably made it much easier to get funding.

In the end, the DENDRAL project did not have much impact on today's computational mass spectrometry. Citing Gasteiger *et al.* [119]: "However, it is sad to say that, in the end, the DENDRAL project failed in its major objective of automatic structure elucidation by mass spectral data, and research was discontinued. Therefore, investigations of the relationships between structure and mass spectra by computer techniques suffered severe setbacks." Possibly, the people behind DENDRAL were too far ahead of their time: With high mass accuracy data and the compute power that we have today, things obviously become easier. Possibly, they chose the wrong objects to study: Peptides are much more convenient, from a computational perspective. As Biemann *et al.* [21] observed back in 1966, "the structures of oligopeptides follow a few strict requirements which can be simply expressed in computer language." But then, rule-based systems have not had much success in peptide analysis. Finally, concepts from theoretical computer science, such as correctness proofs or worst-case running time analysis, did not play a role back then. Still, it might be a good idea for everybody who has a "new" idea on this topic, to look at "ye olde" publications.

10.8 Historical notes and further reading

See Fernie *et al.* [112] and Last *et al.* [202] for introductions to metabolomics and metabolite profiling.

This chapter is somewhat different from the previous ones, in that the computational analysis of small molecule MS has only recently (around 2006) started to flourish. This is somewhat surprising, as the DENDRAL project (Sec. 10.7) started back in 1965, one year before tandem mass spectrometry was invented. In the next four decades, papers were published regularly (say, ten per year) on the "computerized analysis" of metabolite MS data, but little progress was made regarding *general computational methods*. One noteworthy exception is predicting the presence or absence of certain substructures from EI fragmentation data using machine learning [67, 144, 196, 335, 337, 354, 356, 365, 373]. See Scheubert *et al.* [309] for a review. In comparison, computational mass spectrometry for peptides and proteins started only in 1997 or 1999, at least on the (bio)informatics side, but quickly outstripped the development of computational methods for metabolites. I can remember an Informatics session at the annual conference of the American Society of Mass Spectrometry where the chair asked, "who in the room is *not* doing peptides or proteins?", and I was the only one to raise my hand — out of at least 200 people at that session. Maybe, everybody else was shy.

Archetypical methods for the first three approaches to search molecular structure databases are CFM(-ID) [1, 2] for "structure to spectrum", MetFrag [303, 368] and MAGMa [289, 290] for "structure

plus spectrum to score", and FingerID [149] and CSI:FingerID [93, 323] for "spectrum to structure". The Input Output Kernel Regression version of CSI:FingerID [43, 44] avoids predicting a molecular fingerprint, which considerably speeds up computations but comes at the price that, well, we do not get to know the molecular fingerprint. Quantum chemistry methods can be found in [14, 50, 137]. MetFusion [123] is a hybrid approach: Like a kernel method, it computes the similarity of the query spectrum to all spectra in the training data, then combines these similar to a linear support vector machine. But the weights of the linear combination are not learned and instead chosen individually for each candidate, based on the chemical similarity between the candidate and the training structure. Molecular networking was introduced by Watrous *et al.* [363] and is available through GNPS [257, 361]. Examples of "guilt by association" methods are the propagation of structure annotations in molecular networks [68], recursive annotation along a metabolic reaction network [324], and ZODIAC [220], which uses Gibbs sampling to annotate molecular formulas; see also Rogers *et al.* [300] and Sec. 8.9. "Guilt by association" methods that rely on structure databases *may suffer heavily from hidden priors* (Sec. 12.4).

Claude helped!

Searching in molecular structure databases is sometimes referred to as "dereplication", similar to searching in spectral libraries. But this is a rather stupid name, as it implies that we can only find things which are already known. Given that our database can consist solely of hypothetical structures, which for sure nobody "knows" to be biomolecules, it is clear that the label "dereplication" is misleading at best.

In 2008, Hill, Kertesz, Fontaine, Friedman and Grant [151] were the first, as far as I know, who proposed searching in a molecular structure database; they used a commercial program to predict tandem MS spectra from molecular structures. In 2010, Wolf, Schmidt, Müller-Hannemann and Neumann [368] presented a somewhat greedy heuristic to match molecular structures and experimental data, see Sec. 10.5. Methods for searching in molecular structure databases have made remarkable progress since 2008: CASMI (Critical Assessment of Small Molecule Identification) challenges (`http://casmi-contest.org/`) have been conducted since 2012, usually under the (co-)supervision of Emma Schymanski and Steffen Neumann, to evaluate the power of computational methods for searching small molecule tandem MS in structure databases [254, 315, 317]. See there for the progress automated methods have made in only a few years, and see [157–159, 251] for reviews on computational methods for small molecule tandem MS.

We claimed that genome sequencing does not allow us to deduce the structure of the metabolites. This is not true for polyketides, secondary metabolites that are made by polyketide synthases which, in turn, are huge proteins resembling conveyor belt factories. Also, certain proteins are known to make metabolites of a particular structure. Finally, heavily modified and cyclic peptides are at the border of proteomics and metabolomics; as for lipids, it is not advisable to analyze them with general-purpose metabolomics methods.

For the number of molecular structures, see Kerber *et al.* [183] for nominal mass 146, and Kerber *et al.* [185] for masses up to 150. The fact that growth is potentially again exponential, can be "empirically established" from Fig. 1 in [185]. For the crude approximation in (10.2) see Fig. 10.3. Another number that circulates in the literature is "10^{60} molecules with up to 30 atoms of C, N, O, S" [39], often paraphrased as "up to mass 500 Da"; but this estimate is very crude and probably far off the true number.

See Cayley's paper from 1875 [51] on the number of alkanes; but also note the correction by Rains and Sloane [277]. The book chapter by Faulon, Visco Jr and Roe [107] gives an extensive review on the subject, and lists the number of alkanes, alkenes, alkynes, stereoalkanes, ketones, esters, alcohols, benzenoids, and fullerenes. The approximation (10.1) for alkanes is due to Böcker and Wagner [38]. MOLGEN [17, 183, 184] was the state of the art for decades to generate molecular structures for a given mass or molecular formula, but unfortunately, it is commercial. The open-source Open Molecule Generator (OMG) [268] is usually orders of magnitude slower, in particular for large molecular structures where it hurts most. But today, the open-source Surge [237] outperforms MOLGEN by orders of magnitude in speed. As noted, running time differences do not matter that much for computational MS, as ranking millions of structure candidates is extremely challenging. Faulon *et al.* [107] name other computational tools for generating molecular structures given a molecular formula; I have to admit that I do not know which of those are still actively maintained. Finally, Ruddigkeit *et al.* [302] enumerated 166.4 billion molecules with up to 17 atoms from CHNOS plus certain halogens; aggressive filtering was used to cut down the number of molecular

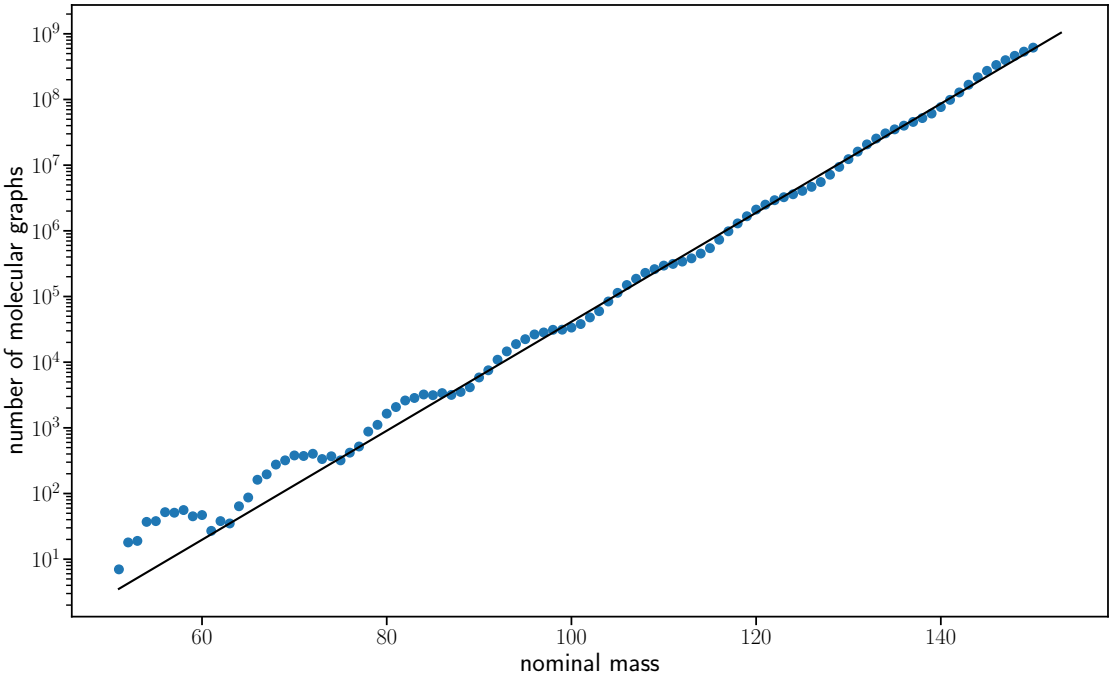


Figure 10.3: Number of molecular graphs with nominal mass m over the alphabet of elements CHNO. Numbers taken from Kerber *et al.* [185]. Note the logarithmic y-axis. Regression line fitted for masses 86 Da and above.

structures based on chemical and biochemical knowledge. Unfortunately, the resulting database GDB-17 is not available as a whole, only subsets of it. Buehler *et al.* [47] went up to 20 atoms of CHNOS plus halogens, estimating that there are about 32 trillion such molecules; using generative artificial intelligence, they sampled 12.1 billion of them for the database GDB-20. GDB-20 is freely available and, hence, the largest freely available database of this kind.[1]

Claude helped!

The work of Heinonen, Rantanen, Mielikäinen, Kokkonen, Kiuru, Ketola and Rousu [147] is targeted at explaining what you see in a tandem mass spectrum of a metabolite, as introduced in Sec. 10.5. Their ILP-based approach suffers heavily from the complexity of the underlying problem, and the resulting combinatorial explosion; running times can be prohibitive even for medium-size molecules. FiD is based on earlier (2006) work by almost the same authors [148], which also contains an NP-hardness proof for the problem of matching masses to substructures. These questions, in turn, go back to earlier work by, again, Hill and Mortishire-Smith [150]. To populate the edge weights of the molecular graph for combinatorial fragmentation, we can use the enthalpy change upon bond fragmentation [222]; smarter ways of computing these weights would be beneficial.

Lacroix *et al.* [199] proposed the Graph Motif problem for searching motifs in metabolic networks. Fellows *et al.* [108] showed that the problem is NP-hard even for bipartite graphs of maximum node degree four and only two colors. Böcker *et al.* [37] evaluated weighted Graph Motif algorithms for cleaving fragments from a precursor molecule, and found that the simple branch-and-bound heuristic performs best for this application, see Sec. 10.5. See Sikora [327] for an overview of theoretical results on the Graph Motif problem; and see Guillemot and Sikora [141] for some parameterized algorithms for different flavors of the problem, and how running times increase when edge weights are taken into account.

Beyond searching in molecular structure databases, *de novo* structural elucidation of small molecules is called "Computer Assisted Structure Elucidation" (CASE). This is usually based on 2-dimensional Nu-

[1] https://gdb.unibe.ch/downloads/

clear Magnetic Resonance (NMR) spectroscopy, and uses mass spectrometry data as secondary data at best. In 2012, Schymanski *et al.* [316] suggested, for GC/EI-MS data, using MOLGEN-MS with substructure classifiers to generate candidate molecular structures, and ranking them by a consensus score of MetFrag, MOLGEN-MS spectral matching, and retention behavior.

LipidBlast is due to Kind *et al.* [191], the Lipid Data Analyzer is by Hartler *et al.* [146], and the rule-based lipid prediction in CFM-ID version 3.0 (which no longer fits the name) is by Djoumbou Feunang [83].

10.9 Exercises

10.1 What happens to sugar-containing metabolites if you heat them? What do scientists do to prevent that, when they want to analyze metabolites by GC-MS?

10.2 Find all subgraphs with molecular formula CHN in the molecular graph of caffeine, see Fig. 10.1.

10.3 Assume that the number of structures with nominal mass m is indeed given by (10.2). Calculate the number of molecular structures with mass *up to* 1000. Hint: You will have to derive the formula for $\sum_i x^i$ or look it up somewhere.

11 Glycan *De Novo* Sequencing

"Sweets for my sweet — sugar for my honey!" (Doc Pomus and Mort Shuman)

Glycans are molecules made from simple sugars that form complex tree structures. Glycans constitute one of the most important post-translational modifications of proteins: Apweiler *et al.* [7] estimate that more than half of all proteins are glycosylated, i.e., carry a glycan modification. Glycans are believed to play an important role in cell growth and development, tumor growth and metastasis, immune recognition and response [278], and even the allergic reaction to white wine [263]. The elucidation of glycan structure remains one of the most challenging tasks in biochemistry. Like metabolites, but unlike proteins, the structure of glycans cannot be directly inferred from the genome sequence of an organism.

One of the most powerful tools for glycan structure elucidation is tandem mass spectrometry. As for peptides, glycan mass spectra can be interpreted by searching a database of glycan structures, but such databases are vastly incomplete.

In this chapter, we focus on the problem of *de novo* interpretation of glycan tandem MS data. This is very similar in spirit to peptide *de novo* sequencing, see Chapter 2. In fact, we will re-use several ideas from earlier chapters, and this chapter can be seen as an application of what we have already learned — with a twist. Note that the term "glycan sequencing" is somewhat ill-chosen, as glycans are trees and not sequences. We will stick with it, though, to avoid word monsters such as "glycan *de novo* topology elucidation", and to underline the analogy to peptide *de novo* sequencing.

There are different levels of resolving the structure of a glycan: We concentrate on the "high-level" structure, namely, the "topology" of the glycan. In some sense, glycan sequencing is more difficult than peptide sequencing, as we try to resolve a tree structure (the topology of the glycan) instead of a linear string. We use a two-step approach suggested in Sec. 2.8, the first step being *candidate generation* and the second step being *candidate evaluation*. As for peptides, we will focus on the candidate generation step. Many early tools for glycan sequencing use a naïve approach to generate candidates: They decompose the precursor mass of the glycan over the alphabet of monosaccharides, then enumerate all topologies that have the correct multiplicities of monosaccharides. This approach faces the problem of a combinatorial explosion of structures, see below. In the following, we will present a smarter way to generate candidates, based on the observed tandem MS data.

Recall that peptide sequencing can be solved in linear time if there is only one ion series, see Exercise 2.1. So, what is the twist here? It turns out that sequencing glycans is computationally hard (NP-hard), even if we simultaneously restrict ourselves to (i) a single ion series and (ii) a simple scoring where each mass contributes -1, 0, or $+1$, penalizing masses without a peak. This can be seen as a late justification for assuming ideal data in Chapter 2: Sequencing glycans is computationally hard, and this hardness does not come from any peculiarities in the mass spectrometry data, but is an intrinsic part of the combinatorial problem itself. If, in turn, we allow peaks to be counted multiple times, the resulting problem can be easily solved, but results can be pathetic, see for example Exercise 11.6.

We have seen in Chapter 9 that an NP-hard problem does not necessarily mean the end of all days. We will again derive a dynamic programming algorithm which not only allows us to find the optimal solution, but also to sample suboptimal ones. And again, the resulting algorithm is fixed-parameter tractable (Sec. 9.4) where, for our theoretical analysis, the parameter k is the "number of peaks in the measured spectrum". In practice, parameter k can be chosen arbitrarily and allows us to tune the methods, trading specificity of the candidate generation for running time and memory consumption.

monosaccharides	mol. formula	mass (Da)
pentoses (Pen), such as xylose (Xyl)	$C_5H_8O_4$	132.042259
deoxyhexoses (dHex), such as fucose (Fuc)	$C_6H_{10}O_4$	146.057909
hexoses (Hex), such as glucose (Glc), galactose (Gal), mannose (Man)	$C_6H_{10}O_5$	162.052823
hexose acids (HexA), such as glucuronic acid (GlcA)	$C_6H_8O_6$	176.032088
N-acetylhexosamines (HexNAc), such as N-acetylglucosamine (GlcNAc)	$C_8H_{13}NO_5$	203.079373
N-acetylneuraminic acid (NeuAc, also Neu5Ac)	$C_{11}H_{17}NO_8$	291.095417
N-glycolylneuraminic acid (NeuGc)	$C_{11}H_{17}NO_9$	307.090331

Table 11.1: Monosaccharides commonly found in glycans, with molecular formula and monoisotopic mass of the residue (removed H_2O). Masses are computed with high mass accuracy, then rounded to six decimals.

Finally, we demonstrate the combinatorial explosion of glycan topologies by counting glycan topologies. In Sec. 11.7, we present methods for counting all glycan topologies with n monosaccharides, and for counting glycan topologies for a given mass m.

11.1 Glycans and glycan topologies

Glycans are — besides nucleic acids and proteins — the third major class of biopolymers, and are built from simple sugars (monosaccharides). Since monosaccharides can have up to five linkage sites, glycans can be assembled in a tree-like structure, making their primary structure considerably more complex than that of proteins. Glycans can be attached to proteins (N-glycans, O-glycans, glycosaminoglycans) or lipids, but may also be free molecules. Starch, glycogen, cellulose, and chitin are sometimes also referred to as glycans, but we will focus on smaller oligosaccharides that usually have more complex structures. Glycosylation, the attachment of glycans to proteins, is presumably one of the most extensive and complex protein PTMs.

Monosaccharides (simple sugars) are the building blocks of glycans. Table 11.1 lists monosaccharides commonly found in higher animals; others can be found in bacteria and plants. A large number of monosaccharides exist, but only a few are present for an individual species or cell: For example, humans cannot synthesize NeuGc because of a missing enzyme [59], although small amounts from dietary sources are incorporated. Molecular formulas and masses in Table 11.1 are reported for monosaccharide residues; for the corresponding monosaccharide, add H_2O or 18.010565 Da. Masses have been computed with high mass accuracy, and *not* with Table 7.1. We see that monosaccharides are mostly made up of carbon, hydrogen, and oxygen, with only a few nitrogen atoms. Note the large number of isomers: Pentoses have five carbon atoms in the backbone, and all share the molecular formula $C_5H_{10}O_5$. Similarly, hexoses with six carbon atoms in the backbone have molecular formula $C_6H_{12}O_6$.

The following paragraph explains how glycans are formed from monosaccharides. As our algorithms deliberately ignore linkage types and only focus on glycan topology, one does not have to understand this paragraph in order to understand the algorithms in this chapter. For us, the important fact is that each monosaccharide has one "in-link" (the anomeric carbon) and usually four "out-links" (carbon hydroxy groups), and that monosaccharides can be glued together via these links (glycosidic bonds). But for the sake of completeness, let us have a look at how glycans are formed from monosaccharides. We will ignore chirality in our presentation, as this does not modify the masses of monosaccharides. See Chapter 2 of [353] for details.

Free monosaccharides can exist in an open or cyclic form, see Fig. 11.1. Carbon atoms are consecutively numbered, starting with the aldehyde carbon atom C-1 double-bonded to an oxygen atom. In glycans, we only find the cyclic form of monosaccharides: Here, the monosaccharide has a ring structure

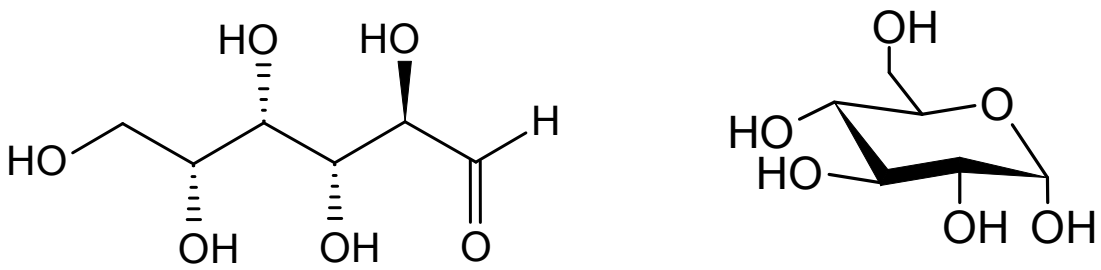


Figure 11.1: Open and cyclic form of the monosaccharide glucose: The chain form of D-glucose (left) and α-D-glucopyranose (right).

with five or six covalent bonds, made from carbon atoms and one oxygen atom. See again Fig. 11.1 (right) for glucose: Atoms C-1 to C-5 plus one oxygen make up the ring of the cyclic monosaccharide. Two monosaccharides can be concatenated by a glycosidic bond, which is formed between the anomeric carbon of one monosaccharide and a hydroxy group of another: Chemically speaking, the hemiacetal group of one monosaccharide reacts with the alcohol group of the other monosaccharide, releasing water. In Fig. 11.1 (right), C-1 is the anomeric carbon, and C-2, C-3, C-4, and C-6 have hydroxy groups. This results in different linkage types, denoted "1–2" for a glycosidic bond involving anomeric carbon C-1 in the first monosaccharide, and carbon C-2 with a hydroxy group in the second monosaccharide. For ketoses such as the sialic acids NeuAc and NeuGc, the anomeric carbon is C-2, resulting in linkages such as 2–3 and 2–6. In a glycan, the unique monosaccharide that is not engaged in a glycosidic bond via its anomeric carbon, may be attached to a protein (see below) or a lipid. This is the distinguished *reducing end* of the glycan: Precisely speaking, "the reducing end of the oligosaccharide bears a free anomeric center that is not engaged in a glycosidic bond and thus retains the chemical reactivity of the aldehyde" [353]. It is still referred to as the reducing end, if the monosaccharide is in fact linked to, say, a serine or threonine.

The following is a rough classification of glycans:

N-glycans (or N-linked glycans) are attached to an asparagine residue of a protein or peptide. The amino acid sequence an N-glycan can be attached to, is either asparagine-X-serine or asparagine-X-threonine, where X is any amino acid except proline. All N-glycans are derived from a common precursor, which is then extensively modified. Still, unlike O-glycans, all N-glycans share a rather similar topology. N-glycans are important for protein folding, among others, and they are very common in eukaryotes but less common in prokaryotes.

O-glycans (or O-linked glycans) are attached to a serine or threonine residue of a protein or peptide. The most common type of O-glycans (mucin-type) starts with an N-acetylgalactosamine monosaccharide. Unlike N-glycans, there is no common precursor, and eight "core structures" are known, of which four are common. O-glycans are very common in eukaryotes but less common in prokaryotes.

Glycosaminoglycans have a linear topology and contain long repetitions of disaccharide motifs. With the exception of hyaluronan, they are attached to a protein, usually via an O-link to serine.

Free glycans are not attached to anything. They are used as signaling molecules for a variety of biological processes, such as plant defense response. Also, free glycans are found in the milk of mammals, and glycans found in human milk appear to protect infants against pathogens affecting the intestines.

So much for the biochemistry; let us come back to computational mass spectrometry, introducing a formal model for glycan topologies. Unlike for peptide sequencing, the alphabet of monosaccharides (the glycan building blocks) can differ depending on the type of glycan we are analyzing. We assume that the alphabet Σ of monosaccharides is fixed and provided by the user, based on the biological background of the experiment. Every element $a \in \Sigma$ is assigned a residue mass $\mu(a)$. At this level, no monosaccharide

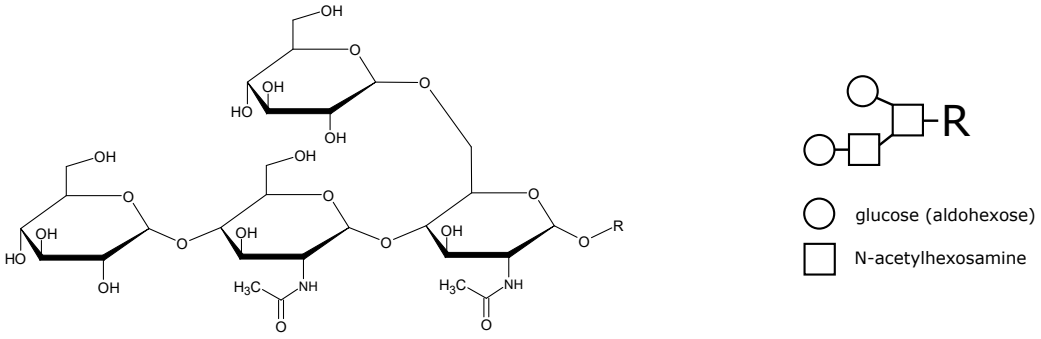


Figure 11.2: Structural formula (left) and topology (right) of a glycan made from four monosaccharides.

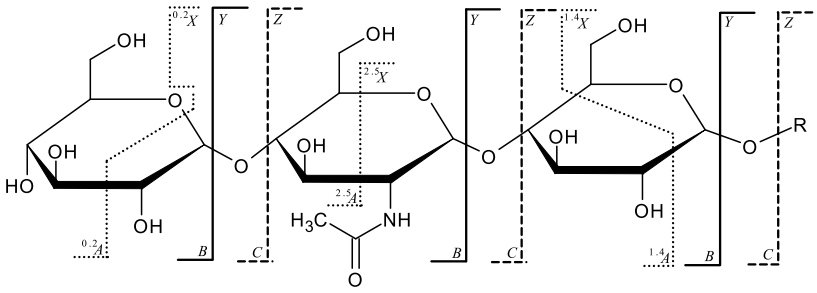


Figure 11.3: Fragments resulting from tandem mass spectrometry analysis of a glycan. Note that B, C, Y, and Z ions are not affected by linkage types.

isomers can be differentiated; depending on our background knowledge about the glycan, we may assume either hexose (Hex) or, say, glucose (Glc) to be part of our alphabet Σ.

We model a *glycan topology* as a rooted tree $T = (V, E)$. The root of the tree is the distinguished root monosaccharide, which can be attached to a protein or peptide. Tree nodes are labeled with monosaccharides from Σ and, hence, each node in the tree is also assigned a mass. Every node has an out-degree of at most four, because each monosaccharide has at most five linkages. See Fig. 11.2 for an example. We will use the letter T to denote both the underlying tree structure, and the glycan topology that includes node labels. To find the molecular formula or mass of a glycan topology we simply add up the molecular formulas or masses of the constituting monosaccharides and, finally, add H_2O or 18.010565 Da. Note that glycan topologies do not contain information regarding linkage types.

Our method will take into account all possible glycan topologies, deliberately ignoring all biological restrictions on, say, the amount of branching in the tree. It is well known that certain branching types are observed seldom in biological samples: For example, most monosaccharides show only one to three linkages, so most nodes in a glycan tree will have out-degree of at most two. But instead of completely forbidding such structures, we can incorporate biological restrictions into our scoring model, by subtracting a penalty if a structural rule is violated. In this way, we do not impede the discovery of rare structures that may diverge significantly from structural restrictions.

11.2 Glycan fragmentation

There are three types of fragmentation that break the glycan topology, resulting in six types of ions, see Fig. 11.3: X, Y, and Z ions correspond to fragments that contain the reducing end of the glycan and are called *reducing end ions* or *reducing end fragments*. A, B, and C ions, in contrast, do not contain the reducing end. A and X ions are cross-ring fragments that result from internal monosaccharide breakages; the exact breakage positions are denoted by an additional superscript.

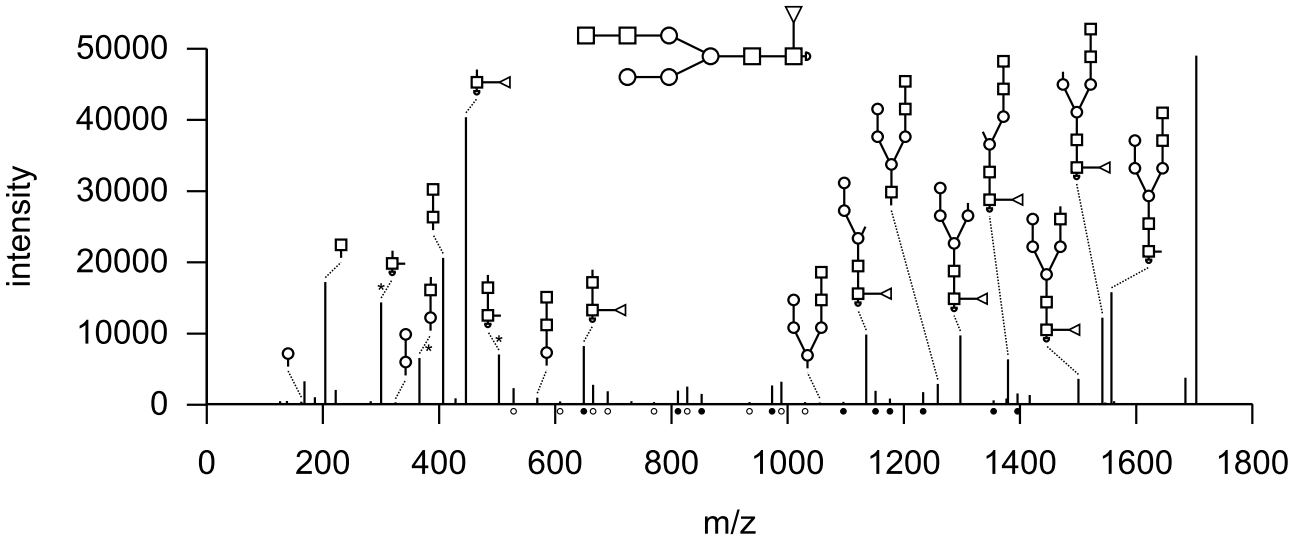


Figure 11.4: The peak-picked spectrum of a glycan with four hexoses, four N-acetylhexosamines, and one fucose. The spectrum is annotated with single-cleaved fragments of the correct glycan topology, and all but one peak corresponding to single-cleaved fragments are found in the spectrum. Three intense peaks marked with (*) correspond to double-cleaved fragments. Additionally, all peaks marked with filled circles can be annotated with double-cleaved fragments of the glycan, and peaks marked with unfilled circles can be annotated with fragments that stem from more than two cleavages. Figure from [29].

In the following, we assume that we have recorded a fragmentation spectrum of a single glycan. For glycans, collision-induced dissociation (CID) is often used as the fragmentation technique. The collision energy determines the collision strength: The higher the energy, the more and stronger atomic bonds break. Since glycosidic bonds between sugars are weak compared to bonds inside the monosaccharides, we can choose the energy so that mainly these bonds break. This will predominantly generate B and Y ions, in particular for protonated precursors, and we concentrate on these two types in our presentation.

We have modeled a glycan topology as a rooted tree $T = (V, E)$, where nodes are labeled with monosaccharides from an alphabet Σ. A fragment T' of T is a connected subtree, and the mass of T' is the sum of masses of the constituting nodes. Let $M := \mu(T)$ be the *precursor mass* of the glycan structure. To simplify our presentation, we ignore mass modifications, such as adding the terminal H_2O group, reducing end modifications, the proton mass, or multiple ion series. As for peptides, these modifications can be easily incorporated into our method, see Sec. 11.5 below. Note that depending on the experimental setup and the glycans we are looking at, reducing end fragments may carry a peptide or peptide part as their "mass modification". This will (usually) make it easier for us to differentiate between the B and Y ion series.

For candidate generation, we restrict ourselves to simple fragmentation events, where only a single glycosidic bond is broken. This is a realistic assumption in practice, see Fig. 11.4: By choosing an appropriate fragmentation energy, we can ensure that intense peaks usually correspond to single-cleaved fragments. Formally, such fragmentation is equivalent to removing a single edge. Hence, we can represent each simple fragmentation event by a node $v \in V$, where the subtree $T(v)$ induced by v represents the non-reducing end fragment, and the remainder of the tree is the reducing end fragment. The resulting non-reducing end fragments have the mass of a subtree of T induced by a node v, denoted $\mu(v)$. For reducing end fragments we subtract $\mu(v)$ from the precursor mass M.

11.3 The candidate generation problem

In this section, we formalize the problem of glycan candidate generation: Given the experimental data, we want to generate a small set of candidate glycan topologies, containing the correct topology. As in Chapter 2, we will use the peak counting score to compare hypothetical spectra with the measured one. A more involved scoring using, say, peak intensities is again considered at a later stage, namely Sec. 11.5 below. Also, we will concentrate on finding the best topology, and generating suboptimal topologies (our candidates) comes "for free", as we will be using dynamic programming once more.

To simplify our presentation, let us assume for the moment that all our mass spectra consist of *non-reducing end ions only*. It turns out that we can easily generalize our solution to include reducing end ions. Also, we assume that all fragments stem from single fragmentation events. For the moment, we assume all masses to be integer.

Assume we are given a glycan topology T, and we want to evaluate T against the measured spectrum. We use a simple fragmentation model to generate a hypothetical *candidate spectrum*, and count the number of shared peaks between the measured spectrum and the candidate spectrum. Let $f(m)$ be the characteristic function of the measured spectrum, telling us if a peak is present (then, $f(m) = 1$) or absent (then, $f(m) = 0$) in the measured spectrum at mass m. Summing $f(m)$ over all peak masses m that are present in the candidate spectrum, we count all peaks that are common to both the measured spectrum and the candidate spectrum. (Again, we can replace this peak counting score by something more elaborate at a later stage.) Formally, we define

$$S(T) := \sum_{m=0,\dots,M} f(m) \cdot g_T(m) \tag{11.1}$$

where

$$g_T(m) := \begin{cases} 1 & \text{if } T \text{ contains some subtree } T(v) \text{ with mass } \mu(v) = m \\ 0 & \text{otherwise} \end{cases} \tag{11.2}$$

is the characteristic function of the glycan topology T, telling us if the tree contains a subtree of a certain mass. The important point is that $g_T(m) = 1$ holds if there is *at least* one subtree of mass m, independent of the *actual number* of such subtrees. Now, the Glycan Candidate Generation problem can be stated as such: Find a glycan topology T^* such that $S(T^*)$ is maximum; and afterward, find all glycan topologies T such that

$$S(T) \geq (1 - \varepsilon) S(T^*)$$

for some fixed $\varepsilon > 0$. This is the set of glycan topology candidates that is passed to the evaluation step of our sequencing algorithm.

Unfortunately, it turns out that finding an optimal topology T^* is an NP-hard problem: Precisely speaking, the decision problem "is there a glycan topology T such that $S(T) \geq t$ for some threshold t?" is NP-hard [322], even if we restrict ourselves to binary trees, where each monosaccharide node has at most two children; the proof uses scores $f(m) \in \{-1, 0, +1\}$, penalizing masses without a peak. So, there is little hope for an algorithm with running time polynomial in M; unless P = NP, no such algorithm can exist. This makes glycan sequencing quite different from peptide sequencing, where this oversimplified version of the problem is comparatively easy to answer, see Chapter 2.

Let $T = (V, E)$ be a glycan topology. We introduce another scoring model, namely

$$S'(T) := \sum_{v \in V} f(\mu(v)) \tag{11.3}$$

which compares the measured spectrum, encoded by the characteristic function f, with a candidate glycan topology. Unfortunately, S' is not a peak counting score. Instead, for *every* subtree T' of T with mass $m' = \mu(T')$ we add $f(m')$ to the score. In this way, a glycan topology that contains many subtrees of identical mass m' receives a high score if $f(m')$ is large, even if it ignores all other peaks; see for example

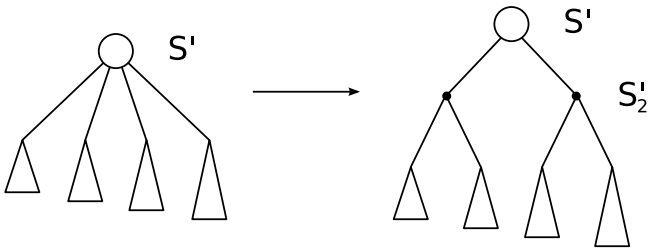


Figure 11.5: In (11.4) we compute the score for appending up to four previously computed subtrees to one monosaccharide, the root of the current subtree. Equation (11.5) reduces the complexity of computation by appending two "headless" subtrees to the root monosaccharide; each "headless" subtree, in turn, consists of two subtrees. Subtrees can be empty.

Exercise 11.6. We will show in the next section how computations for this model can be modified to avoid peak double counting, though.

To find the glycan topology T that maximizes the score $S'(T)$, we define $S'[m]$ to be the maximum score of any glycan topology with total mass m. We assume $S'[m] = -\infty$ if there is no glycan topology of that mass. It is easy to see that S' can be computed by the recurrence

$$S'[m] = f(m) + \max_{m_1+m_2+m_3+m_4+\mu(a)=m} S'[m_1] + S'[m_2] + S'[m_3] + S'[m_4], \tag{11.4}$$

where the maximum is taken over all $a \in \Sigma$ and $0 \le m_1 \le m_2 \le m_3 \le m_4 < m$; see Exercise 11.2. The term $S'[m]$ corresponds to any subtree of mass m with an arbitrary monosaccharide at its root. We initialize $S'[0] = 0$, as the "empty glycan topology" does not explain any peaks. We further assume $S'[m] = -\infty$ for all $m < 0$. But what about monosaccharide nodes that do not have the maximum out-degree of four? Actually, these are already covered in (11.4): If one or more of the m_j in (11.4) equals zero, then the monosaccharide at the root of the subtree has fewer than four bonds. The maximum score of any glycan topology of precursor mass M is $S'[M]$.

Unfortunately, computation of $S'[M]$ takes much too long using (11.4), see Exercise 11.3. Luckily, we can speed up computations considerably:

$$\begin{aligned} S'[m] &= f(m) + \max_{a\in\Sigma} \max_{m_1=0,\dots,\left\lfloor \frac{m-\mu(a)}{2} \right\rfloor} S'_2[m_1] + S'_2\big[m-\mu(a)-m_1\big] \\ S'_2[m] &= \max_{m_1=0,\dots,\lfloor \frac{m}{2} \rfloor} S'[m_1] + S'[m-m_1] \end{aligned} \tag{11.5}$$

The term $S'_2[m]$ corresponds to a "headless" subtree without a monosaccharide at its root, see Fig. 11.5. See Exercise 11.4 regarding the correctness of this recurrence. Now, instead of attaching four children to a monosaccharide node, we attach two headless subtrees with two children each. Remember that the special case of fewer than four children is covered in (11.4), as one or more subtrees can have mass zero, and it is also covered here.

Using (11.5) we can compute $S'[M]$ in time $O(|\Sigma| \cdot M^2)$: We have to compute M entries in S' and S'_2; computing $S'[m]$ requires $O(|\Sigma|\, M)$ time, and computing $S'_2[m]$ requires only $O(M)$ time. The actual algorithm is simply a For-loop over all masses $m = 0, \dots, M$; we omit the simple details. It should also be understood how to recover an optimal solution using backtracing, see Exercise 11.5. We reach:

Lemma 11.1. *Given a monosaccharide alphabet Σ with masses $\mu : \Sigma \to \mathbb{N}$, a precursor mass M, and a function $f : \{0, \dots, M\} \to \mathbb{R}$ encoding the measured spectrum. Then, we can compute $S'[m]$ and $S'_2[m]$ for all $m = 0, \dots, M$ in time $O(|\Sigma| \cdot M^2)$ using recurrence (11.5). Next, we can recover a glycan topology $T = (V, E)$ maximizing $S'(T)$ in $O(|V| \cdot |\Sigma| \cdot M)$ time, backtracing through S' and S'_2.*

11.4 An exact algorithm for glycan candidate generation

So, the problem of generating glycan topologies from tandem MS data is NP-hard, even if we restrict ourselves to the simple peak counting model — what can we do? For obvious reasons, we want to stick with the dynamic programming approach, as this allows us to generate an arbitrary number of suboptimal solutions (Sec. 2.8). Also for obvious reasons, we do not want to rely on heuristics at this early stage of our algorithm: If the true solution is missed during candidate generation, it cannot be brought back during candidate evaluation.

We now modify recurrences (11.5) to find the glycan topology T that maximizes $S(T)$. We do not want to take a closer look at the proof of NP-hardness in [322]; but I can assure you that the complexity of the problem only holds for mass spectra that contain a "large" number of peaks. Measured spectra, in contrast, are relatively sparse and contain only tens of peaks that have significant intensity: The number of simple fragments of a given glycan topology corresponds to the number of nodes in a tree, and this is only linear in the number of constituting monosaccharides. Let k be the number of peaks in the measured spectrum: k is the parameter of our problem, and we limit the running time explosion to this parameter, while maintaining a polynomial running time with respect to M. Choosing k to be *equal* to the number of peaks, is solely done for the ease of presentation. In the next sections, we show that parameter k can be arbitrarily chosen in practice, trading specificity for running time and memory consumption of the method: For low k, the method produces more candidates that have a high score because of scoring peaks multiple times. As done previously in this textbook, we abandoned optimality to save time and space. It turns out that a moderate k, such as $k = 10$, is appropriate in practice [29].

In order to avoid multiple peak counting, we incorporate the set of explained peaks into the dynamic programming. Let C be the set of peak masses in the measured spectrum, where $|C| = k$; these will serve as our *colors*, see Chapter 9. For every mass $m \leq M$ and every subset $C \subseteq C$ we define $S[C, m]$ to be the maximum score of any glycan topology T with total mass $\mu(T) = m$ where only the peaks from C are used to compute this score. We stress that we do not have to use all peaks from C, and that it is OK to only score a subset. At the end of our computations, $S[C, M]$ holds the maximum score of any glycan topology where at most the peaks from C are taken into account for scoring. We initialize $S[C, 0] = 0$ for all $C \subseteq C$.

We could come up with a recurrence similar to (11.4) for computing $S[C, m]$ but this would be much too slow in practice, see Exercise 11.7. Instead, we modify the faster recurrence from (11.5) for our purpose: We define $S_2[C, m]$ to be the score of a "headless" glycan topology with mass m using only peaks in C. Again, we initialize $S_2[C, 0] = 0$ for all $C \subseteq C$.

Now, S_2 helps us to restrict the branching in the tree to bifurcations: We limit the recurrence of $S[C, m]$ to two "headless" subtrees with disjoint peak sets $C_1, C_2 \subseteq C$, where C_1 is the subset of peaks explained by the first subtree, and C_2 is the set of peaks explained by the second subtree. We require $C_1 \cap C_2 = \emptyset$ which guarantees that every peak is scored at most once. Additionally, we demand $C_1 \cup C_2 = C \setminus \{m\}$. We obtain the following recurrences:

$$\begin{aligned} S[C, m] &= \max_{a \in \Sigma} \max_{m_1 = 0, \ldots, \left\lfloor \frac{m - \mu(a)}{2} \right\rfloor} \max_{C_1 \subseteq C \setminus \{m\}} \Big\{ f(C, m) + S_2[C_1, m_1] \\ &\qquad + S_2\big[C \setminus (C_1 \cup \{m\}), m - \mu(a) - m_1\big] \Big\} \\ S_2[C, m] &= \max_{m_1 = 0, \ldots, \left\lfloor \frac{m}{2} \right\rfloor} \max_{C_1 \subseteq C} S[C_1, m_1] + S\big[C \setminus C_1, m - m_1\big] \end{aligned} \tag{11.6}$$

What is $f(C, m)$? We have to delay the scoring of a peak at mass m if m is not in C: To this end, we define

$$f(C, m) := \begin{cases} 0 & \text{if } m \notin C \text{ but } m \in C, \\ f(m) & \text{otherwise.} \end{cases} \tag{11.7}$$

So, both peaks not in C and peaks in C are scored, whereas scoring of peaks in $C \setminus C$ is delayed. For the particular case that C equals the set of all peaks (or peak colors), the condition "$m \in C$" is always fulfilled. But as we have indicated above, we want this recurrence to work for an *arbitrary* set of peak colors.

```
1:  procedure GlycanSequencing(mass M, set of peak colors C)
2:      Initialize S[C, 0] ← 0 and S_2[C, 0] ← 0 for all C ⊆ C
3:      for m = 1, ..., M do
4:          for all subsets C ⊆ C do
5:              Compute S[C, m] using (11.6)
6:          end for
7:          for all subsets C ⊆ C do
8:              Compute S_2[C, m] using (11.6)
9:          end for
10:     end for
11: end procedure
```

Algorithm 11.1: Generating glycan candidates: Besides the precursor mass M and the set of peak colors C, the weighted alphabet Σ with integer masses $\mu : \Sigma \to \mathbb{N}$ and the function $f : \{0, \ldots, M\} \to \mathbb{R}$ are inputs of the method.

We do not want to prove the correctness of recurrence (11.6), as this proof will be somewhat lengthy and technical. We just note that the presumably best way to prove the correctness, is to show the equivalence of (11.6) to a recurrence for four subtrees similar to (11.4), and then to show that this recurrence does indeed compute the correct values $S[C, m]$. See Exercise 11.8.

It is not very involved to compute (11.6) in an admissible order, see Algorithm 11.1. To see this, take a closer look at (11.6): Computing $S[C, m]$ accesses table entries $S_2[\cdot, m_1]$ with $m_1 < m$ but never for $m_1 \geq m$; furthermore, it does not access any entries $S[\cdot, \cdot]$. So, we ensure that $S_2[C, m_1]$ has been computed for all $C \subseteq C$ and all $m_1 < m$, which is trivial, see the outer loop of Algorithm 11.1. Similarly, computing $S_2[C, m]$ will only access entries $S[\cdot, m_1]$ for $m_1 \leq m$, but not $S_2[\cdot, \cdot]$. So, we ensure that $S[C, m_1]$ has been computed for all $C \subseteq C$ and all $m_1 \leq m$, which is again trivial: We simply interleave the computation of $S[\cdot, \cdot]$ and $S_2[\cdot, \cdot]$.

We now analyze time and space that is needed for the computation of (11.6). One can easily see that the space required to store $S[C, m]$ is $O(2^k \cdot M)$. What is the time required to compute all $S[C, m]$? It turns out to be $O(3^k \cdot |\Sigma| \cdot M^2)$. To understand this, we first note that factors $|\Sigma|$ and M come from the calculation of $S[C, m]$ in (11.6), and that we can ignore computation of $S_2[C, m]$ in our considerations. As we have to compute $S[C, m]$ for all $m = 0, \ldots, M$, another factor M is obvious. But we have to be careful about partitioning the set C: There are 2^k sets $C \subseteq \{1, \ldots, k\}$, and iterating over all $C_1 \subseteq C$ takes $O(2^k)$ time, resulting in a 4^k factor in our running time analysis. But this analysis is simply too imprecise: There exist 3^k possibilities to partition k peaks into the three sets C_1, C_2, and $C \setminus (C_1 \cup C_2)$. Every such partition will be evaluated exactly *once* in the computation of $S[C_1 \cup C_2, m]$, so *any* algorithm that computes (11.6) in a reasonable fashion, and in particular Algorithm 11.1, has the desired running time of $O(3^k \cdot |\Sigma| \cdot M^2)$.

To recover an optimal solution, we backtrace through the dynamic programming matrix starting from entry $S[C, M]$. It is obvious that the maximum score will explain as many peaks as possible, but not necessarily all. We can also compute suboptimal solutions that deviate at most $\delta := \varepsilon \cdot S(T^*)$ from the optimal score $S(T^*)$ for some $\varepsilon > 0$, see Sec. 2.8. This backtracking usually generates many isomorphic trees, which we have to remove from the final output: This can be achieved by encoding glycan topologies as strings, see Exercise 11.9. Running time of backtracking is $O(out \cdot 2^k \cdot |\Sigma| \, M n)$ where n is the maximum size of a glycan topology in the output, and *out* is the number of generated trees including isomorphic trees, which is usually larger than the size of the final candidate set.

11.5 Mostly old wine in new skins

In this section, we discuss modifications to the algorithm from the previous section, many of which are meant to make it run fast in practice. Most of these modifications are already known to us from the problem

of peptide sequencing and the rest of this textbook — hence, the title of this section. We will refer to ideas and tricks throughout this textbook, which might be slightly uncomfortable for the reader, but illustrates the degree to which we can "recycle" these ideas.

It is understood that we can use a more elaborate scoring than the peak counting score; in particular, we can use real-valued masses for the scoring (Sec. 8.8), and peak intensities and peak masses as described in Secs. 4.7 and 4.8. Only when incorporating peak intensities, the presented approach can be applied to experimental data.

Certain glycan topologies do in fact create the same fragment mass several times: It must be understood that $S(T)$ does not *penalize* such topologies, but it also does not *reward* them. The latter is important: As the extreme case, consider a single leaf of the tree. If the corresponding peak has very high intensity, then any score that does not prevent double counting would reward trees for having identical labels at all leaves, which is surely not desirable.

Next, we get rid of the unrealistic assumption that only non-reducing end ions (B ions) are present in the mass spectra. But whenever we find a non-reducing end ion then, for ideal data, the complementing reducing end ion must also be present in the mass spectrum. We can "mirror" the spectrum in a preprocessing step, see Sec. 2.5.3. The spectrum now contains reducing end and non-reducing end peaks with the same intensity for every observed peak, even if only one was detected by the instrument. But how do we avoid multiple peak counting for intense peaks? Some glycan topology might contain both a B and a Y ion of identical mass, and if a peak with this mass is present in the measured spectrum, we must not score it twice. In fact, avoiding multiple peak counting comes "for free": We regard the elements in C as colors, and assign complementing reducing end and non-reducing end ion peaks the same color. This can be achieved without changing recurrence (11.6) or Algorithm 11.1. This ensures that each peak is only scored once, either as a reducing end ion, or as a non-reducing end ion, whatever leads to the better score. In practice, this problem is usually not as daunting as for peptide *de novo* sequencing: The mass of a reducing end ion is often not decomposable if regarded as non-reducing end ion mass (and vice versa) because of reducing end modification, and will not be considered for the score anyway. This can be attributed to the small monosaccharide alphabet, where many masses cannot be decomposed, see also below. Recall that in some cases, the mass modification of the reducing end can be a peptide or peptide fragment. Here, the complementing ion series, such as B and Y ions, will show only small overlap, further simplifying the problem.

We can incorporate C and Z ions into our score by merging their intensities with those of the corresponding B and Y ion pair. By this, we are losing our "peak counting score" interpretation of the optimization, but this is most probably not a problem in applications.

We do not penalize for additional peaks that are not explained by our glycan, see Sec. 4.6 for a justification. Recall that for candidate generation, this is a sensible thing to do, as our scoring does not take into account peaks from multiple-cleaved fragments, or the A/X ion series. For our computations, this implies that we can safely ignore additional peaks, and we have to compute $S[C,m]$ and $S_2[C,m]$ only for those sets $C \subseteq \mathcal{C}$ that do not contain any additional peaks/colors. This can be efficiently implemented by sparse dynamic programming using hash maps to store values, and by restricting (11.6) to initialized entries, see Sec. 15.5. The maximum score is no longer found in entry $S[\mathcal{C},M]$ which is now usually undefined; instead, we search for the maximum entry $S[C,M]$ with $C \subseteq \mathcal{C}$, and we have to start backtracing from this entry. When computing (11.6), sets C containing peaks with mass larger than the current mass m need not be considered.

In case the number of peaks in a glycan mass spectrum is too large, we can easily limit the exponential growth in memory and running time by choosing an appropriate k such as $k = 10$. Now, we use the k most intense peaks C in our explanation at most once, whereas we allow all other peaks to contribute multiple times to the score.

Using these engineering techniques, the prohibitive factor[1] in the running time may easily become M^2. But we noted above that many masses m are not decomposable at all over the alphabet of monosaccharide masses, as our alphabet Σ is usually small in applications (Exercise 11.1). Using sparse dynamic programming, we can altogether exclude these masses from our computation, since there exists no subtree which could explain them. A similar argument shows that the mass remainder $M - m$ must also be decomposable over the alphabet of monosaccharide masses, so we can also exclude many masses close to M from our computations. Doing so, we again reduce running time and memory requirements of the algorithm in practice.

We can also include a priori probabilities for the number of branches a monosaccharide has. For example, fucose generally does not connect to further monosaccharides. These probabilities can be estimated from known glycan structures. All recurrences presented in this chapter and, in particular, (11.5) and (11.6) can be modified to take into account properties of the monosaccharide a, such as the number of links of a for the scoring: This is obvious for the "long" recurrence (11.4) but can also be achieved for the recurrences using "headless" subtrees, see Exercise 11.10. Be aware that care has to be taken when learning branching probabilities: A particular branching may be rare in general but common at a certain position of the glycan, and the scoring must not impede such branching. As an example, we mention mannose in the core structure of certain N-glycans, with out-degree three.

11.6 Evaluation of glycan topology candidates

Once we have reduced the set of potential glycan topologies from the exponential number of initial candidates, to a manageable set of tens or hundreds of structures, we can now evaluate each candidate glycan topology using an in-depth comparison between its theoretical spectrum and the measured spectrum. This comparison can also take into account multiple-cleaved fragment trees, C and Z ions, or A and X ions where a monosaccharide is cleaved. Furthermore, we can now penalize missing peaks when we expect to see a B or Y ion.

But as indicated in Sec. 4.11, this is again not the best we can do. Clearly, if we observe C and Z ions (which are assumed to have relatively low intensity) then we should also see the (high-intensity) B and Y ions; if this is not the case, our interpretation may be incorrect, so we should penalize the candidate. More involved is the following reasoning: If we observe a peak that our scoring wants to attribute to a double-cleaved fragment, it is reasonable to assume that we should also see a single-cleaved fragment it originated from; if this fragment does not exist, our hypothesis may again be wrong. The same is true for multiple-cleaved fragments.

Finally, we can take into account prior biological knowledge on the glycans we are considering: In particular, N-glycans, O-glycans, glycosaminoglycans and free glycans have certain preferred "branching patterns". As always, it is not a smart idea to filter out "unexpected" results; rather, we penalize them. As existing glycan databases are presumably vastly incomplete and do not contain information about the frequency of glycans, we should not derive too detailed priors from there: For example, few monosaccharides are connected to four or even five other monosaccharides; but if we transform database counts into a prior, we have made it practically impossible that such a glycan will ever receive the top score.

11.7 Counting glycan topologies

We have mentioned above that the number of glycan topologies easily becomes prohibitive for enumerating all possible topologies. We now substantiate this claim, by computing the number of glycan topologies for a certain number of monosaccharides, and for a certain mass. Our analysis will be strictly combinatorial, as we will ignore whether geometrical constraints render certain glycan topologies impossible: These

[1] The "prohibitive factor" in a running time is the one that will make the problem "crack" in practice, meaning that we have to wait hours (days, years) for the solution.

glycan topologies will show a large amount of branching, particularly close to the root. In this sense, the numbers computed here are upper bounds to the "true" number of glycan topologies; but these bounds are "sharp" in the sense that, from a combinatorial standpoint, our calculations will be exact. Taking into account considerations from molecular geometry will be extremely difficult: This might boil down to enumerating all glycan topologies using the methods presented below, then testing for each glycan topology if it can exist in 3D space. On the other hand, our computations do not take into account linkage types or chirality.

Let $r^{(s)}[n]$ be the number of different glycan topologies with n nodes, where nodes are labeled with elements from Σ and $s = |\Sigma|$. One can easily see that this number does not depend on the actual alphabet Σ but only on its cardinality.[2] Recall that a glycan topology corresponds to a rooted tree such that every node has out-degree at most four. In the following a *glycan tree* is a rooted tree with out-degree at most four. Note that counting glycan trees does not take into account that nodes are labeled, and corresponds to the case $|\Sigma| = 1$.

The "classical way" of counting trees, is to use Pólya's enumeration theorem [274]. We omit the details, but you eventually come up with the recurrence

$$\begin{aligned} r[n] = \frac{1}{24}\Bigg(& \sum_{i+j+k+l=n-1} r[i]r[j]r[k]r[l] + 6 \sum_{i+j+2k=n-1} r[i]r[j]r[k] \\ & + 3 \sum_{2i+2j=n-1} r[i]r[j] + 8 \sum_{i+3j=n-1} r[i]r[j] + 6 \sum_{4i=n-1} r[i] \Bigg) \end{aligned} \tag{11.8}$$

where $r[n] := r^{(1)}[n]$ is the number of glycan trees, and we initialize $r[0] = 1$. Similar to Sec. 3.7, one can approximate the number of glycan trees $r[n]$ using a closed formula: This number asymptotically behaves like

$$r[n] \sim 0.4621373461 \cdot n^{-3/2} \cdot 2.911037772^n. \tag{11.9}$$

We omit the details, but see also below.

We can roughly estimate the number of glycan topologies over an *arbitrary* alphabet Σ by $|\Sigma|^n \cdot r[n]$, since every node can be colored with an individual color. This overestimates the number of glycan topologies, as we do not take into account isomorphic trees; see Exercise 11.11. As an example, consider $|\Sigma| = 5$ and $n = 10$: We estimate this number to be $6.28 \cdot 10^9$ whereas the true number is $3.10 \cdot 10^9$, so the true number is only *half* of what our rough estimate tells us. The relative error will become even larger as n increases, see Exercise 11.12.

We now present a method for the exact computation of $r^{(s)}[n]$ for an alphabet Σ of size s. Somewhat surprisingly, the resulting formulas are not very complicated and computations are very fast; but unfortunately, the math behind these formulas are highly non-trivial and require an understanding of generating functions. For this reason, we simply present the formulas and numbers, but do not explain *why* these calculations are correct. If you are interested in that part, see Böcker and Wagner [38].

The formula to calculate the exact number $r^{(s)}[n]$ of glycans of size n over an alphabet Σ of size $s = |\Sigma|$, is

$$\begin{aligned} r^{(s)}[n] = s \cdot \Bigg(& \frac{1}{24} r_4[n-1] + \frac{1}{4} \sum_{k=0}^{\lfloor (n-1)/2 \rfloor} r[k] r_2[n-1-2k] + \frac{1}{8} \sum_{k=0}^{(n-1)/2} r[k] r[(n-1-2k)/2] \\ & + \frac{1}{3} \sum_{k=0}^{\lfloor (n-1)/3 \rfloor} r[k] r[n-1-3k] + \frac{r[(n-1)/4]}{4} \Bigg) \end{aligned} \tag{11.10}$$

[2] We will not use k to denote the size of the alphabet in this section, as we have "consumed" it for the parameter k in the previous sections.

n	approx. using (11.9)	number of glycan topologies $\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert = 2$	$\lvert\Sigma\rvert = 3$	$\lvert\Sigma\rvert = 4$	$\lvert\Sigma\rvert = 5$
1	1	1	2	3	4	5
2	1	1	4	9	16	25
3	2	2	14	45	104	200
4	4	4	52	246	752	1 800
5	9	9	214	1 485	5 996	17 850
6	19	19	904	9 369	50 288	186 750
7	44	45	4 038	61 947	440 784	2 039 500
8	105	106	18 508	421 668	3 980 384	$2.30 \cdot 10^7$
9	257	260	87 008	2 939 562	$3.68 \cdot 10^7$	$2.64 \cdot 10^8$
10	639	643	416 388	$2.09 \cdot 10^7$	$3.46 \cdot 10^8$	$3.10 \cdot 10^9$
15	72 669	72 917	$1.25 \cdot 10^9$	$4.51 \cdot 10^{11}$	$3.07 \cdot 10^{13}$	$8.24 \cdot 10^{14}$
20	9 866 949	9 881 527	$4.51 \cdot 10^{12}$	$1.16 \cdot 10^{16}$	$3.23 \cdot 10^{18}$	$2.61 \cdot 10^{20}$
25	$1.48 \cdot 10^9$	$1.48 \cdot 10^9$	$1.78 \cdot 10^{16}$	$3.29 \cdot 10^{20}$	$3.75 \cdot 10^{23}$	$9.08 \cdot 10^{25}$
30	$2.35 \cdot 10^{11}$	$2.35 \cdot 10^{11}$	$7.50 \cdot 10^{19}$	$9.89 \cdot 10^{24}$	$4.62 \cdot 10^{28}$	$3.36 \cdot 10^{31}$

Table 11.2: Number of glycan topologies for n nodes and an alphabet size of one to five, plus rounded approximation (11.9) corresponding to $|\Sigma| = 1$.

where

$$r_2[n] = \sum_{k=0}^{n} r[k]r[n-k], \quad r_3[n] = \sum_{k=0}^{n} r[k]r_2[n-k], \quad r_4[n] = \sum_{k=0}^{n} r[k]r_3[n-k]. \tag{11.11}$$

Note that different s also require different $r_2[n]$, $r_3[n]$ and $r_4[n]$; we have written $r_i[n]$ instead of the more precise $r_i^{(s)}[n]$ solely for the sake of readability. In (11.10), we set $r[x] := 0$ if x is not an integer. It is easy to check that we can compute $r^{(s)}[n]$ in $O(n^2)$ time and $O(n)$ space. See Table 11.2 for the number of different glycan topologies for an alphabet size of one to five.

But generating functions allow us to do more: We can also approximate the number $r^{(s)}[n]$ of glycans over an alphabet of size s. In detail, we reach:

$$\begin{aligned} r^{(1)}[n] &\sim 0.4621373461 \cdot n^{-3/2} \cdot 2.911037772^n \\ r^{(2)}[n] &\sim 0.4359963877 \cdot n^{-3/2} \cdot 5.603311188^n \\ r^{(3)}[n] &\sim 0.4286144321 \cdot n^{-3/2} \cdot 8.305741452^n \\ r^{(4)}[n] &\sim 0.4251715830 \cdot n^{-3/2} \cdot 11.01087067^n \\ r^{(5)}[n] &\sim 0.4231864914 \cdot n^{-3/2} \cdot 13.71711627^n \end{aligned} \tag{11.12}$$

See [38] for details. Note that the approximation of $r^{(1)}[n] = r[n]$ has already been reported in Eq. (11.9) and is included here for completeness. These approximations are very accurate even for small n, see Table 11.2: For $n = 10$ we reach an approximate value of $r^{(1)}[10] \approx 638.6$ using Eq. (11.12) whereas the true number is $r^{(1)}[10] = 643$, so the relative error is well below one percent. What we can learn from (11.12) is that we are indeed facing a combinatorial explosion of candidates, as the number of glycan topologies is increasing exponentially in n; furthermore, the exponential growth is already rather steep even for small alphabets: For alphabet size 5 we have roughly 13.7^n glycan topologies, compared to 5^n strings.

Last but not least and somewhat surprisingly, a similar recurrence to Eqs. (11.10) and (11.11) is possible for glycan topologies over Σ with a particular mass M; this number can be computed in $O(|\Sigma|\, M^2)$ time and $O(M)$ space [38]. As an example, we report the number of glycan topologies over integer-weighted alphabets of size 1 to 3 in Fig. 11.6. We again observe strong combinatorial effects: For most masses of up to 3000 Dalton we do not find any glycan topology. This is again because most masses cannot be decomposed over the monosaccharide alphabet, see Sec. 8.6. A nice feature of the recurrence

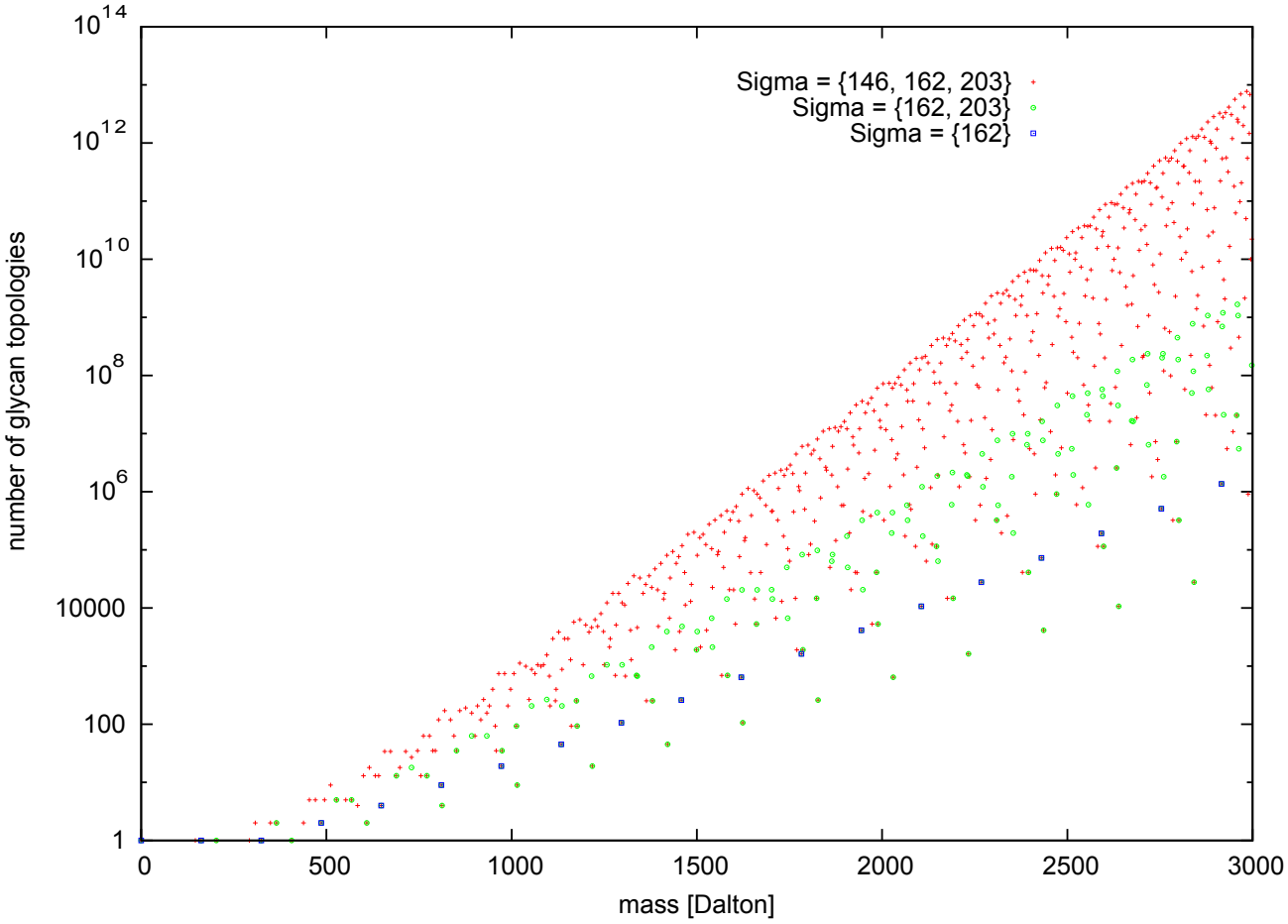


Figure 11.6: Number of glycan topologies for varying mass M and the monosaccharide alphabets $\Sigma = \{162\}$ (hexose), $\Sigma = \{162, 203\}$ (hexose, N-acetylhexosamines), and $\Sigma = \{146, 162, 203\}$ (fucose, hexose, N-acetylhexosamines). Y-axis (number of topologies) is logarithmic. Whenever no point is present in the plot, this implies that there is no glycan topology of the corresponding mass. Figure taken from [38].

for computing glycan topologies of mass M, is that we can also use it to *enumerate* all glycan topologies of mass M, similar to what we did for strings in Sec. 5.2. If you want to enumerate glycan trees with a fixed number of monosaccharides, though, then there are faster and simpler ways [212].

One might argue that the number of glycan topologies that we can find in biology, will be much smaller, as we rarely observe that any monosaccharide links to five other monosaccharides. To do so, we can modify calculations to count glycan topologies with maximal out-degree three. Again, we can approximate the number ${}^{(3)}r$ of glycan trees with out-degree at most three using a closed formula,

$$ {}^{(3)}r[n] \sim 0.5178760 \cdot n^{-3/2} \cdot 2.815460^n. \tag{11.13} $$

Finally, if you feel that out-degree three might still be too high for biologically relevant glycans, here is the approximation for the number of glycan trees with out-degree at most two:

$$ {}^{(2)}r[n] \sim 0.791603 \cdot n^{-3/2} \cdot 2.483254^n. \tag{11.14} $$

We see that the growth is still exponential, and that the *base of the exponential growth has not decreased substantially* between out-degree four to three: Namely, the base decreased from 2.911 to 2.815.

11.8 Historical notes and further reading

See Raman *et al.* [278] for an introduction to the field of glycomics. The book "Essentials of Glycobiology" by Varki *et al.* [353] can be accessed freely over the Internet.[3]

Many approaches for glycan *de novo* sequencing follow the two-step approach (Sec. 2.8) of first generating candidates, then scoring them [106, 121, 127, 322, 342]. The following is a sample of methods published until 2008: STAT by Gaucher *et al.* [121] first decomposes the masses of all peaks found in the spectrum; the user has to manually decide on one of the decompositions. STAT then generates all glycan topologies for the chosen decomposition, and uses certain restrictions to rule out some topologies. StrOligo by Ethier *et al.* [106] enumerates biologically plausible N-glycan topologies for the given precursor mass, and evaluates each topology based on the measured spectrum. OSCAR by Lapadula *et al.* [200] is special in that it does not use MS^2 but MS^n fragmentation data as input. It is a true *de novo* approach and does not require prior biological information. GLYCH by Tang *et al.* [342] not only derives the topology of the glycan, but also deduces linkage types using cross-ring ion fragments. GLYCH uses dynamic programming to compute the optimal score of a structure, similar in spirit to S' from (11.3). The drawbacks of this approach have been described above: Certain peaks might be scored many times by the dynamic programming, leading to somewhat arbitrary optimal solutions. Sheng *et al.* [325] concentrate on solely inferring linkage types. CartoonistTwo by Goldberg *et al.* [127] focuses on O-glycans, which are structurally more diverse than N-glycans. The program uses biological constraints for O-glycans, and scores candidates by an elaborate scheme. And finally, there is GlycoMaster by [322], GlycoDeNovo by Hong *et al.* [154], the deep learning method CandyCrunch by Urban *et al.* [351], and potentially many more programs.

Regarding candidate generation, the above programs can be subdivided into three categories: Some approaches enumerate all possible glycan topologies of the given precursor mass [106, 121, 127]. This is possible as in practice, the alphabet of monosaccharides is usually very small (three to five monosaccharides) and, hence, the number of decompositions is also small — quite often, there is only one decomposition. If glycans become larger, this again results in a combinatorial explosion of tree structures, see Sec. 11.7. To cope with this problem, tools apply strict biological rules to cut down on the number of candidates. GLYCH uses dynamic programming similar to Sec. 11.3 but simply ignores the problem of multiple peak counting [342]. Finally, Shan *et al.* [322] present a heuristic that avoids peak double counting. Regarding scoring the candidates, the by far most elaborate approach is due to Goldberg *et al.* [127], who use dependencies between the observed fragments to modify the score.

Besides the (*de novo*) interpretation of tandem MS data, many other computational approaches have been developed for glycan MS analysis [52, 64, 128–130, 166, 215, 228, 369].

The term "glycan sequencing" has been repeatedly used in the literature for the structural elucidation of glycans; other terms include "extracting sequence information" [121], "glycan structure determination" [127], "glycan structure elucidation", and "glycan structural assignment" [106].

The presentation in this chapter largely follows the paper by Böcker, Kehr and Rasche [29]. Shan, Ma, Zhang and Lajoie [322] introduced the simple scoring model from (11.3) as well as the efficient recurrence (11.5), and established that generating glycan topology candidates while avoiding peak double counts is an NP-hard problem. The scoring from Sec. 11.6 generalizes ideas of Goldberg *et al.* [127].

Böcker *et al.* [29] evaluated the presented approach (computing $S(T)$) for generating glycan topology candidates using 24 glycan spectra. Glycans were composed of fucoses, hexoses and N-acetylhexosamines and ranged in mass from 1379 to 2354 Da. Candidate generation required on average 4 seconds (including trace back) setting $k = 10$. In all but two cases, the correct glycan topology was contained in the generated list of candidates, which usually contained 100 to 200 (and never more than 450) topologies. In comparison, there exist millions of glycan topologies for some of the precursor masses. Avoiding peak double counting is indeed important to generate a useful candidate list: If peak double counting was ignored

[3] https://www.ncbi.nlm.nih.gov/books/NBK579918/

(computing $S'(T)$), then even a candidate list with 500 or more topologies contained the correct structure in only 8 cases.

See Zaia [376] for a review on mass spectrometry analysis of glycans. The nomenclature of ion series is due to Domon and Costello [84].

Similar to Chapter 9, the running time of our dynamic programming algorithm in Sec. 11.3 can be reduced from $O(3^k \cdot |\Sigma|\, M^2)$ to $O(2^k \cdot k^2\, |\Sigma|\, M^2)$ [24]. Again, the practical use seems to be very limited due to the required overhead.

Counting the number of glycan topologies in Sec. 11.7 presents some of the results from Böcker and Wagner [38], which contains more results on counting (node-labeled) trees of bounded degree. Approximations (11.9) and (11.13) are due to Otter [260], who also showed how to approximate these numbers for arbitrary fixed out-degree d. Unfortunately, he left out the constants for rooted trees with maximal out-degree four, see (11.9). Note that the multiplicative constant of (11.14) is correct [115, page 477], but different from the one in [260]. The number of glycan topologies with $|\Sigma| = 1$ or, equivalently, the number of rooted trees where each node has out-degree at most four, is sequence A036718 in the On-Line Encyclopedia of Integer Sequences.[4] If you want to find approximations for pretty much any type of trees, take a look at Flajolet and Sedgewick [115] and Harary *et al.* [145]. See also Exercise 11.14.

Li and Ruskey [212] show how to enumerate trees with bounded degree, see Sec. 2.2 of their paper. The amortized running time per tree is "constant for realistic values of n", such as $n \leq 25$, where n is the number of nodes in the tree.

11.9 Exercises

11.1 Compute the number of decompositions over the alphabet Pen, dHex, Hex, HexA, and HexNAc using mass accuracy 0.001 Da (multiply masses by 1000, round down). Compute and plot minimum, median and maximum for superbins of width 1 Da as in Figures 8.2 and 8.3 for the mass range 0 to 2500 Da.

11.2 Prove that recurrence (11.4) is correct, that is, $S'[m]$ computed using this recurrence is truly the maximum score of any glycan topology with total mass m.

11.3 Establish the running time of computing $S'[M]$ using (11.4).

11.4 Show that computations using (11.4) and (11.5) will lead to identical results.

11.5 Show how to recover an optimal solution from the array S' using backtracing. What is the running time?

11.6 Assume that we have two monosaccharides $\Sigma = \{\mathsf{F}, \mathsf{H}\}$ with masses $\mu(\mathsf{F}) = 146$ and $\mu(\mathsf{H}) = 162$. Assume further that we have recorded the mass spectrum (without ion series modifications)

$$\mathcal{M} = \{146, 162, 308, 454, 746\}$$

with precursor mass $M = 746$. Assume further that the peak at mass 146 has intensity 2, whereas all other peaks have intensity 1. Assume that we consider peak intensities in the scores $S(T), S'(T)$ as proposed in Sec. 11.5; so, the peak at mass 146 scores twice as much as the other peaks. What is the glycan topology maximizing $S(T)$; and what is the glycan topology maximizing $S'(T)$?

11.7 What is the running time of computing $S[C, m]$ by a recurrence similar to (11.4)? Note the base of the exponential growth!

★ 11.8 Prove the correctness of recurrence (11.6). You can do so by first showing its equivalence with a recurrence similar to (11.4), see Exercise 11.7; then, showing that this recurrence will compute the correct values.

[4] http://oeis.org/A036718

11.9 Describe an algorithm to transform a glycan topology into a string, that *uniquely* describes the glycan topology. The important point here is that for glycan topologies, the children of a monosaccharide are unordered, and we have to give them a "canonical ordering."

11.10 It is obvious that we can score the "degree of branching" in (11.4), by counting the number of m_i with $m_i = 0$. It is slightly less obvious how to achieve this for (11.5) and (11.6) while at the same time, guaranteeing the optimality of the computation. Discuss the problem for (11.5), and show how it can be modified accordingly. Hint: If $S'_2[m]$ is optimal for $m_1 = 0$, then $S'_2[m] = S'[m]$.

11.11 We noted in Sec. 11.7 that $r^{(|\Sigma|)}[n] \approx |\Sigma|^n \cdot r[n]$ overestimates the true number of trees. Show, for $n = 5$ and $\Sigma = \{\mathsf{F}, \mathsf{H}\}$, what glycan topologies are counted by this estimate, and which of them should not be counted (multiple times).

11.12 Compare the "rough estimate" $|\Sigma|^n \cdot r[n]$ to the exact number, and calculate the relative error, for $|\Sigma| = 1, \ldots, 5$ and $n = 1, \ldots, 30$.

⋆ 11.13 Use the recurrence from Böcker and Wagner [38] to enumerate all glycan topologies of mass M for the alphabet $\Sigma = \{132, 146, 162, 176, 203\}$ and sub-alphabets of your choice.

★ 11.14 The constants in approximations (11.13) and (11.14) have only six decimal places. Find more accurate constants, as described in point VII.21 on pages 477–479 of Flajolet and Sedgewick [115]. Alternatively, you can use the 20-step recipe from Harary *et al.* [145].

12 Priors and Machine Learning: Overfitting and Self-Fulfilling Prophecies

"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts." (Sherlock Holmes, A Scandal in Bohemia)

"With four parameters I can fit an elephant, and with five I can make him wiggle his trunk." (John von Neumann, according to Enrico Fermi)

Priors, the stochastic equivalent of any "prior knowledge", are a double-edged tool: On the one hand, they are sometimes indispensable to make sense of the data; on the other hand, they can easily result in self-fulfilling prophecies and overrating the power of a computational approach. I have stumbled upon such badly used prior knowledge repeatedly in computational mass spectrometry, so I believe that some words of warning are in order. (Be warned that only in the vast minority of cases, using priors is mentioned as such.) Machine learning, on the other hand, is now widely used in science, and increasingly so in computational mass spectrometry. Again, I have stumbled upon severe problems repeatedly, some of which are closely related to the problems of using priors.

In this (short) chapter, I will describe a few suggestions on and warnings about using priors and machine learning. This will be sometimes in the form of examples, and these examples will often focus on small molecule MS. This *does not mean* that other areas of algorithmic mass spectrometry do not have exactly the same problems! It is simply owed to the fact that my research focuses on small molecules; all of the examples are from experiences that my group and I have had during the last years. Feel free to come up with similar examples for, say, proteomics. I will switch back and forth between prior knowledge and machine learning, because some of the warnings cannot be attributed to either machine learning or using prior knowledge. I am not an expert on either subject, so do not expect any deep insights. But I have been surprised how often even this "shallow advice" is not taken into account.

As an initial word of warning: You must not feed a large number of mass spectra into machine learning, and then expect it to sort things out for you. A lot of people believed that this is possible, and to the best of my knowledge, nothing sensible came out of that. In an infamous example, the authors of some study (not to be named here) directly fed raw SELDI (surface-enhanced laser desorption/ionization) MS data to machine learning, without even picking peaks. They found that the five most important features to discriminate between healthy and sick individuals, are at mass over charge 0.42, 0.08, 0.07, 0.43, and 0.05. The authors reached 97 % correct classifications using the ten top-ranked values, nine of which have m/z below 1. Considering that a single proton weighs more than 1 Dalton, this finding has to be interpreted with some care.

And another word of warning: Machine learning methods suffer from all issues mentioned in this chapter, not only those where it explicitly says "machine learning suffers". The problem is that machine learning methods assume that the training data is a representative sample of the data that the method will later be applied to. This assumption is almost always violated in practice; *your training data is rarely a representative sample*! There are numerous reasons why people measure particular things, but generating representative samples is rarely the intention. (The millions of peptide reference tandem mass spectra that have been measured in different labs around the world, may be one of the first examples where people tried to generate a representative sample: But even there, I assume that heavily modified peptides are rare, potentially because these are more expensive to synthesize. Money, by the way, is often a good explanation of what is measured and what is not.) Consequently, you are involuntarily introducing priors through the

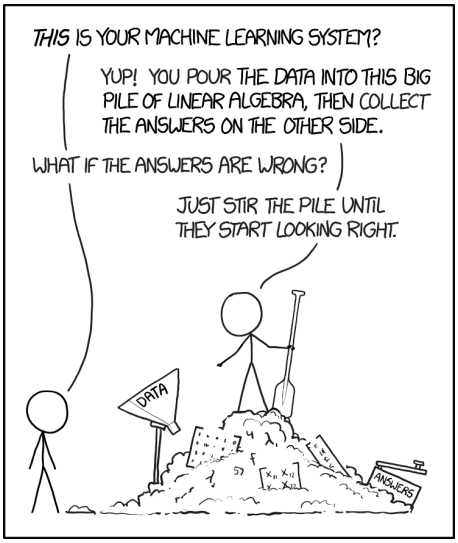


Figure 12.1: Machine learning by xkcd (https://xkcd.com/1838/).

training data that you use. To avoid nasty surprises, you will have to search for these priors before you can train. Have fun! 😉

A final comment may seem in order: This chapter is so full of "don't do this" and "don't do that", that you as a reader may wonder, "what is wrong with this guy?" In my defense: Each of the warnings in this chapter is based on one, and very often more than one, publications I had the pleasure to read.

12.1 Priors and prior information

In Bayesian statistical inference, a prior is the probability distribution that would express one's beliefs about some quantity before evidence, such as a tandem mass spectrum, is taken into account. Bayes' theorem implies that

$$\mathbb{P}(M \mid D) = \frac{\mathbb{P}(D \mid M) \cdot \mathbb{P}(M)}{\mathbb{P}(D)}$$

where $\mathbb{P}(M \mid D)$ is the *posterior probability* (the conditional probability of the model given the data), $\mathbb{P}(D \mid M)$ is the *likelihood* of the data (the conditional probability of the data given the model), and $\mathbb{P}(M)$ is the *prior probability* (the probability of the model *before* any data is taken into consideration). We usually cannot compute the probability of the data $\mathbb{P}(D)$ but this is often not a problem: We simply normalize posterior probabilities so that they sum to one, over all possible models; alternatively, we search for the model M that maximizes the numerator $\mathbb{P}(D \mid M) \cdot \mathbb{P}(M)$.

There has been a long and vivid discussion about the justifications of "Bayesian inference" vs. "frequentist inference". This is none of our business; for bioinformatics, Bayesian inference has been extremely successful. A classical textbook example is a rapid test for, say, a human immunodeficiency virus (HIV) infection: Such a rapid test may have specificity of 99 % (out of 100 people without an HIV infection, the test is negative for 99 people) and sensitivity (recall) of 99 % (out of 100 people that have an HIV infection, the test is positive for 99 people). But if the infection is rare (say, only 0.01 % of the tested individuals have an HIV infection) then it is much more likely that someone with a positive test result *does not* have an HIV infection, see Exercise 12.1.

When I speak about "priors" in this chapter, I am not referring to the distinction between Bayesian inference vs. frequentist inference. Instead, I mean any *prior information* that we use in our calculations and that is not directly in the data that we are analyzing. By "prior" I simply refer to integrating prior knowledge into your optimization or calculation. In this sense, a scoring matrix used to calculate a pairwise

sequence alignment is prior knowledge — frequentists may blush when they hear me speak like this, and I hope I am not causing any heart attacks. As said, I am not a statistician.

After reading this chapter, some readers might say, "how lucky am I that I have not chosen a probabilistic framework with all these prior problems, but rather a no-nonsense score where this cannot happen!" But the truth is:

- A prior is a prior even if you do not know what a prior is.

12.2 Pirates in the Caribbean

Be aware that with prior values to be selected, you are *introducing additional parameters* into your model. Simply by doing so, you can make sure that results get better in evaluation. This is completely independent of the fact whether or not the prior actually made any sense. For example, you can use the number of pirates in the Caribbean in a certain decade as a prior for your data. Chances are high that, just by chance, this number is correlated to some number which indeed gives your method an advantage in evaluations, if we are allowed to select an arbitrary value. This does not mean that it made any sense to prioritize on the number of pirates. To avoid this type of overfitting, two suggestions are in order:

- Choose your priors *independently* of the task that you actually want to perform. In particular, do not choose priors so that some performance evaluation number (the *objective function*) is optimized.
- If this is not possible and you have to rely on your objective function, do (10-fold) cross-validation, and also evaluate the resulting method on an independent dataset, see below.

For the first point, consider PAM (point accepted mutation) scoring matrices used for computing protein alignments. Before PAM, people used *ad hoc*-chosen scoring matrices based on biochemical knowledge, which somehow put a score on the similarity or dissimilarity of each pair of amino acids. When Dayhoff, Schwartz and Orcutt [74] developed the PAM scoring matrices, they brought prior knowledge into sequence alignments, which they derived from data: closely related protein sequences, arranged in 71 evolutionary trees. But to do so, they did not choose the 210 free parameters in the amino acid scoring matrix such that, say, a maximum number of positions were correctly aligned for some reference alignment dataset. Instead, the parameters were chosen so that the observed data — closely related proteins with at least 85 % sequence identity — were best explained by a stochastic model of sequence evolution. Similarly, the priors for fragmentation tree computation (Chapter 9) were directly derived from statistical estimates of the data, without considering any optimization criteria such as "molecular formula identification".

12.3 Master of the obvious

Let us assume you did everything right: You want to use some background information as a prior; you determined the parameters of the distribution for the prior without evaluating your method; and to evaluate if this was a good decision, you used a decent cross-validation and truly independent data, as described below. It worked, everybody is happy. But I am a grouch: I still do not believe in your prior.

A simple example that I have used before [26], is a smartphone app for identifying birds from photos. You do some pretty cool stuff and come up with reasonable identification results. Then, someone approaches you with the smart idea to use prior information on how common birds are. You know from background information (Wikipedia) that the great spotted woodpecker is about 10 million times more common than the ivory-billed woodpecker (Fig. 12.2). But if you integrate this prior information into your app, you will rarely and possibly never again identify an ivory-billed woodpecker with your app. The prior is too strong and whatever the data is, this will not be enough to support the ivory-billed woodpecker identification. Nevertheless, the app will suddenly perform much better in evaluation: All these borderline cases where the app previously decided for an ivory-billed woodpecker will now be classified "great spotted





Figure 12.2: Left: Great spotted woodpecker, adult male. Right: Ivory-billed woodpecker, male specimen in Carnegie Museum of Natural History. Left picture by Usitea, license CC BY-SA 3.0 https://creativecommons.org/licenses/by-sa/3.0; right picture by James St. John, license CC BY 2.0, https://creativecommons.org/licenses/by/2.0.

woodpecker", and that is in almost all cases the correct answer. On the other side, the app is now simply telling you things that you already know. If you are an ornithologist, you will be unhappy with how this new and better app is performing; sorting out the bogus ivory-billed woodpeckers is relatively easy; missing one is fatal. In machine learning terms: The prior is correct, but the objective (accuracy) does not reflect the cost of missing a rare bird.

Another helpful example is the aforementioned rapid test for HIV. Using Bayes' theorem we have seen that even if the test result is positive, it is nevertheless much more likely that the tested patient does not have an HIV infection. But it does not make sense that the doctor sends home the patient, telling him "Bayes' theorem tells us that you do not have an HIV infection!" Instead, this would be the time to do other, potentially more expensive tests that have better sensitivity and specificity. The indication that a rare event just got 100-fold more likely, is important information which should not be left aside because of prior information.

It is easy to integrate prior information into many approaches described in this book. For example, you can use the frequency of amino acids in a protein database as prior information for peptide *de novo* sequencing. You may not want to stop at that and use frequencies of di-amino acids in the database, or maybe tri-amino acids. Taken to the extreme, your *de novo* sequencing turns into a database search. There is a fine line between tuning your method for better performance and turning it into a "master of the obvious" that simply repeats what we already know.

One possibility to avoid becoming a "master of the obvious" is to use a flat prior for all "normal" hypotheses, and to use smaller priors only for things that are really, really different from everything we know today. This advice will usually not help you much if you are searching in a database: Everything in a database is there (or at least, should be there) for the reason that this is indeed considered a possibility. In Theodore Woodward's terms, all database entries are horses; what is *not* in the database are the zebras, plus many horses we are currently not aware of. For example, both the great spotted woodpecker and the ivory-billed woodpecker are birds in your database, and should receive a flat prior. But to penalize molecular formulas which are very different from every single (bio)molecule we know today (Sec. 8.4), we may indeed use a prior; we do so based on the assumption that these molecular formulas are also very different from every (bio)molecule we will get to know in the next 50 years. In short:

- Penalize the outlandish, but do not reward the obvious.

Clearly, your prior must not be, "if it is not in some database, it gets a penalty"! Databases are just a snapshot of what is known at this point of time, and will often grow over time. Instead, you will have

to derive properties such that all database entries fall on one side, but examples not in the database fall on the other.

As an example from bioinformatics where people are decidedly *not* using prior information, consider phylogenetics: After decades of research, people are still using maximum likelihood or Bayesian inference with uninformative priors when building phylogenetic trees. This allowed the discovery that whales and dolphins are closely related to hippos and, more broadly, to cows, which was in clear contrast with everything that everybody believed to know at that time. If only they had used prior information to prevent such nonsense discoveries...

But wait, there is more that can go wrong with integrating prior information! And that has to do with the fact that we can rarely evaluate our method on true biological data.[1] If we use a prior which improves your results in evaluation, this may not carry over to biological data; in fact, you may have made everything worse. This might be hard to imagine for someone from proteomics, where labs have started to measure hundreds of thousands of synthetic peptides, resulting in millions of spectra; but these have not been around for long, and other areas of computational mass spectrometry will never have the luxury of such huge and mostly unbiased data for training and evaluating our methods. Consider metabolite identification: You can use the cost at which you can buy a standard, as a prior for metabolite identification. This will work great in evaluation: The spectral library you evaluate on, will rarely contain standards which are extremely costly. But why should metabolites in biological samples follow the same prior probability? Also, what about all the metabolites which you can currently *not* buy, for example because they are only *hypothetical* at present?

So, what is the solution to all of that? Well, I cannot give an easy answer to this question. The only advice I can offer, is:

- Be careful with prior information.

If you are not sure if you are doing the right thing, that is possibly because you are not doing the right thing.

12.4 Hidden priors

Another dangerous situation is when you believe that you are using a flat prior, but in truth, you are not. In particular, a flat prior on one entity might lead to a non-flat prior for another one. See Velasco [355] for a detailed discussion on why the flat priors used in phylogenetic reconstruction are, loosely speaking, not as flat as we might think.

A "textbook example" of a hidden prior is methods that use networks *as an input* for their analysis. (The important differentiation is whether the network is derived from the data, or if it is prior information.) For example, you might want to use a metabolite network when identifying metabolites in your sample: Your candidates (nodes of the graph) are metabolites in a structure database, and you connect these candidates via biotransformations (edges of the graph). These biotransformations may be obtained from some database such as KEGG [171]; alternatively, you may assume that two candidates that have a sufficiently similar structure (or even molecular formula) are connected via an edge. Now, you may give a bonus if many metabolites your metabolite is connected to, are simultaneously present in the same sample.

This sounds reasonable but unfortunately, you have just introduced a strong prior. Clearly, a node that does not have any incident edges, cannot have any active neighbor nodes. Similarly, a node that has few incident edges has on average fewer active neighbors than one with many incident edges. Depending on how you score active neighbors, you either penalize nodes for having neighbors, or you give them a bonus on average. You can now estimate which candidates will be chosen with what prior probability, without

[1] No rule without exception: Methods for FDR estimation in shotgun proteomics are often evaluated on biological datasets. But the information derived from such evaluations is usually only that "FDR estimation method A accepted more peptides at FDR 1 % than method B", without knowing if the estimate by method A is indeed better. Possibly, the additional peptides are all wrong; possibly, method B is more accurate in its estimates. See also Sec. 6.7.

having seen the data. It is understood that to ensure a flat prior, candidates can get a bonus *or a penalty*, depending on the number of active neighbors.

One may argue that a positive prior for candidates with many outgoing connections is justified. (I would object.) But even in case you believe this prior is fine, you have to make it transparent: A user must be able to see all prior probabilities, to make up his or her mind whether these are the priors (s)he believes in. In reality, the situation is even worse: The molecular networks we use are just a snapshot that reflects our current knowledge (and possibly not even that) about biology and biochemistry; they must not be confused with the biological truth. The prior can also lead to inflated numbers in evaluations: Your method might not be doing anything useful but rather pick the usual suspects; see again Sec. 12.3 on why that is a bad idea.

A second example of a "hidden prior" — again from metabolomics, I have to excuse myself — is the identification of molecular formulas, see Chapters 8 and 9. With the advent of methods for searching in molecular structure databases (Chapter 10), some authors observed that even if a method was unable to identify the correct structure, it often got the molecular formula right. Candidates were chosen using the monoisotopic mass, but no additional information was used. This perceived good performance is again due to a prior hidden in the searched structure database, and in the data used for evaluation: For the latter, we use reference compounds; these usually have "common molecular formulas", meaning that there are many other structures in the database with the same molecular formula; which means that the correct molecular formula is chosen correctly in the vast majority of cases even if we use random data as input. So, the perceived performance has little (if anything) to do with the method used for searching in the structure database; you might as well roll a die. The latter can be a simple yet powerful way of detecting such "hidden priors": Rolling dice should not get you anywhere; but if it does, you should proceed with care.

The most reasonable way to deal with this problem is to design your computational method in a way so that you can *prove* that all candidates have a flat prior — and by that, I mean a mathematical proof, not an empirical one. Otherwise, strange things can happen that you never considered when designing the method.

12.5 Cross-validation

In statistics, cross-validation is used to assess how a method will perform in practice. It requires that we partition the sample data into batches, in most cases ten. In ten iterations, the method is trained on nine of the ten batches, and evaluated on the tenth batch. Compared to splitting the data once into training and test data, cross-validation has the advantage that we average results over ten iterations, making our estimates more robust and avoiding "lucky punches". There are machine learning models that are far too large to allow a cross-validation evaluation, as training requires months of computation on very large computers; but unless you are in that business, there is no reason to prefer a single training vs. test split over a cross-validation. This is particularly the case if your machine learning model is quickly trained; reporting only one such split always raises an eyebrow, as it allows us to select a split where our performance metrics made our method shine in the way everyone wants their methods to shine.

When do you have to perform a cross-validation? The answer is simple: As soon as you make one decision (what parts of the scoring should be switched on or off?) or optimize one parameter, then this is training, and there is no way around cross-validation. And if you use cross-validation to make these decisions, then the performance you report must come from a second, outer cross-validation (*nested* cross-validation) or from independent data. In other words: "Training" is not limited to high-end machine learning methods. "Playing around" with a score to improve results *is training, too.* The fact that cross-validation is a "must" and not a "nice-to-have" feature, is unfortunately not generally known in the computational mass spectrometry community. This has resulted in sometimes absurdly inflated numbers to be reported in publications. In short:

- Whenever you tweak or optimize your score, do cross-validation.

And while we are at it: If you are doing *feature selection* as part of training a machine learning model, then this feature selection must be part of the cross-validation setup! If you do the feature selection on all data, then proceed with the machine learning on the selected features, you can actually use random data and come up with excellent model performance. A way around it is to select the features completely independently of the machine learning task at hand, see the previous section.

12.6 Data leakage

Consider the classification of animals from pictures: This is the "*Drosophila*" of machine learning, as it is relatively easy to produce huge amounts of high-quality training data; millions of pictures are available on the Internet, and classifying them is easy for humans. Now, assume that a certain photographer has not uploaded single pictures, but rather multi-shot mode pictures. In this case, we have not one but, say, ten pictures of the same animal; these pictures are not identical but extremely similar. If we now sort these ten pictures into the ten cross-validation batches, or into training vs. test data in case cross-validation is impossible, we no longer have to classify (learn) pictures; it is now sufficient to memorize them.

Another example, this time from computational mass spectrometry: Machine learning is increasingly used to identify small molecules from fragmentation spectra, or to predict fragmentation spectra for a given molecular structure, see Chapter 10. It is understood that mass spectra from the same molecule have to be sorted into the same batch, to prevent the method from simply memorizing these spectra; this is despite the fact that these are independent measurements. Unfortunately, this is not enough: Different stereoisomers of a certain molecule can exist; fragmentation spectra of stereoisomers are not indistinguishable, but *extremely similar*. If we sort stereoisomers into different batches, our method can again memorize these spectra, and evaluation results will be excellent. We can only prevent this if we perform a *structure-disjoint* evaluation: All molecules with the same molecular graph are sorted into the same batch.[2] The above issue is called "*data leakage*" in machine learning: The model has information about the test data (say, a mass spectrum from exactly this structure) that will not be available in practice. The result is "spurious associations" between features and the quantity we want to predict. The situation gets even worse for GC-MS mass spectrometry, where not only spectra of a compound, but also spectra for numerous chemical derivatives are recorded in a library, see next.

Evaluating machine learning models using a scaffold-disjoint cross-validation or training vs. test data split, has become common practice in virtual screening and related areas of drug discovery. Here, the *scaffold* of a molecular structure is its core structure after removing, say, hydroxy groups and other side chains. In a similar direction, Gerlich and Neumann [123] excluded all compounds from the reference spectral library that were structurally similar (instead of identical) to the query compound. With that, you can beautifully observe how the power of the method diminishes when the similarity threshold is decreased. Both approaches may sound exaggerated but in fact, the situation is even worse than what I described above: Large spectral libraries often contain measurements for a compound, plus tens of derivatives or structural analogs of the compound with small modifications, such as replacing a hydrogen by a hydroxy group. The reason is that synthesizing certain derivatives and structural analogs of available compounds is relatively cheap. For GC-MS, the situation is rather extreme, in that derivatization is often part of the experimental setup; consequently, electron ionization fragmentation spectral libraries contain every structure scaffold in possibly thousands of small variations. Derivatives and structural analogs are indeed different structures and often have different molecular formulas than the original compound; unfortunately, they often fragment "mostly the same". Sorting them into different cross-validation batches, you are again memorizing, not learning. In short:

- Make sure that "basically identical" examples are always sorted into the same cross-validation batch. Otherwise, your method is memorizing, not learning.

[2] You can decide if two molecules are identical on this level using the first block of the InChI (International Chemical Identifier) key.

I am aware that this suggestion is in contradiction to the usual race for better evaluation numbers; but as a side effect, you might see that the difference to evaluation on independent data becomes smaller, see the next section. Finally, I want to mention the problem that a molecular graph is often not an adequate representation of the molecular structure — check out "mesomerism" in Wikipedia — but I leave it at that. (The rabbit hole goes deeper and deeper.)

Unfortunately, it often requires a rather deep understanding of the application and the available training data to spot all memorizing and data leakage issues. Let us consider a search engine; and allow me to assume it is, again, a search engine for small molecules (Chapter 10). What we want to do is *rank* candidate molecular structures, based on a query tandem mass spectrum. It is a rather extreme ranking task where only one answer is correct (top rank) and all others are incorrect and, hence, incomparable (any other rank will do); but as we cannot guarantee the correct answer to end up at the top position, we want to rank it as highly as possible. Now, "ranking" sounds like we may consult the rich machine learning literature on "learning to rank", which has been successfully used in information retrieval and internet search engines. So, we present our model no longer only the pairs "molecular structure plus mass spectrum". Instead, for any given query mass spectrum, we now present molecular structures of both the correct answer and of incorrect answers for this query. With this innocent-looking modification of the learning task, we have just introduced a massive case of data leakage: Assume we are using all molecular structures from PubChem (or a comparable, large molecular structure database). For a moment, let us forget about the mass spectrometry information; it is not easy to make sense of, anyway. What can we do with the pairs "molecular structure plus correct/incorrect label"? In fact, that information is very helpful: It turns out that all our training data is from biomolecules (molecular structures of biological interest), and there are several orders of magnitude fewer such structures than all of PubChem. Can we maybe differentiate between biomolecules and the rest of PubChem? In fact, that is pretty simple, and "shallow" machine learning models for doing so are available. Hence, if I were a machine learning model, I would try to predict, "is this structure likely to be one for which MS/MS data have been measured?" Filtering the PubChem candidates down to biomolecule structures, then randomly selecting one of the biomolecules will result in a search performance that is basically on par with even the best methods for searching molecular structure databases, as long as we do our evaluation on PubChem. As it turns out, we do not even need the tandem mass spectrum to reach this performance! That is excellent news, and millions of dollars can be saved on experiments. But then, we did not search in PubChem because it makes sense to do that in practice, but only because it appeared to be reasonably hard for an evaluation. By giving this information to the machine learning model, we have basically destroyed the evaluation setup.

12.7 Cute dogs and smart horses

There exist numerous examples in the literature, showing how the bias in your training data can result in overestimating the power of your method, and even result in a method useless in practice. Some examples make it into the press (photos of Black people were labeled as "gorillas"; I have to apologize for writing this, but that is how it was), but most go unnoticed; the latter are by far the more dangerous. A particularly cute example is the bias in animal pictures: It turns out that Tibetan terriers are usually photographed from the front as a centered close-up, with their eyes at a certain position and distance. I assume that most pictures of Tibetan terriers were uploaded to the Internet by people who wanted to demonstrate this cuteness to everybody; the right posture (front picture, large and tilted head) underlines the message. In comparison, warthogs are rarely taken pictures of in this posture. Now imagine that you train your classifier on this data, but test it on data which is taken by an automated camera, then an animal looking into the camera — preferably with the head tilted — will be classified as "Tibetan terrier" whereas an animal in a different posture will be classified as "warthog". The point here is that you have not trained a classifier for animals; you have trained a classifier for "animal pictures uploaded to the Internet".

Another upsetting example from computer vision is that the presence of a horse in a picture can be inferred from the presence of a "source tag" (photographer name, Internet address) in the lower left corner

of the picture; a car with a source tag is clearly a horse, to the machine learning method at least. This has been referred to as "clever Hans effect" [201].[3] Warnings about such clever Hans predictors are almost as old as machine learning; but with each new hype cycle in machine learning, these warnings are again forgotten. Cues in the mass spectrometry data that machine learning might focus on are, for example, "bleeps" in the raw data that provide information about the machine the sample was measured on, see the example in the introduction of this chapter. Another possibility is that the data was preprocessed by different software tools; or, different samples were treated with slightly different experimental protocols. All of this is hard to spot for a human, but rather simple to pick up for machine learning.

Much like the trainer of clever Hans was entirely unaware he was providing cues which allowed the horse to "sidestep" the problem, many people training machine learning methods are unaware they provide such cues in the training data. Recall that the SELDI example from the beginning of this chapter is, in all likelihood, just a smart horse.

- Do your best to avoid training a smart horse.

12.8 Independent data

The most important point about independent data, is that you must not "touch it" when developing your method and optimizing your parameters. The data are "burned" (no longer independent) as soon as you include it in your loop of developing the method, or selecting the best method. This includes basic decisions such as selecting which prior information your method will use at all, or selecting which machine learning technique (support vector machines or deep neural nets?) you want to apply. Only when you have made your final decision on what your method is and what the parameters are, are you allowed to touch the independent data. This is an ideal and usually out of reach in practice; but you must try hard to get as close as possible to this ideal.

In machine learning, a common approach is to set aside a portion of the training dataset (say, 30 %) before starting method optimization; this subdataset is then sometimes called "independent data". But this is not the case: The randomly selected subset has exactly the same biases etc. as the data you have trained on. This means that you will largely overestimate the power of your method for data that are truly independent. For an MS dataset to be independent, it should have been measured by a different person, in a different lab, on a different instrument, using different samples! If one or more of these conditions are violated, there is a good chance that you reward your method for overfitting to the peculiarities of the data. In short:

- Make sure that your independent data is truly independent.

As for cross-validation, "independent data" in itself is often not enough: You again have to make sure that "basically identical" examples are not simultaneously present in training and independent data. For the example of small molecule fragmentation mass spectra, make sure that no molecules with identical molecular graph are present in the independent data. This will require that you discard a huge number of examples from either the training or the independent data, which is not desirable for obvious reasons: You usually do not have enough data to start with. A better approach is to test, for each example in the independent data, if a "basically identical" example is present in the training data; if so, use the model trained in cross-validation where the "basically identical" training examples are sorted into the testing batch. All of them should be in one batch, see above.

- Use the appropriate cross-validation model even when evaluating on independent data.

[3] Clever Hans ("der Kluge Hans") was a horse that was claimed to have performed arithmetic and other intellectual tasks. This was later shown to be an artifact: The horse was responding to involuntary cues in the body language of his trainer. The trainer was entirely unaware that he was providing these cues.

12.9 Building a skyscraper from a bag of bricks

With the hype around deep learning and deep neural networks [204], people have started to apply them basically for any learning problem. The hype of deep learning is basically built on three pillars: First, new network architectures such as convolutional neural networks [205] and long short-term memory networks [152] are tailored to the structure of the data, so that convolutional networks have far fewer parameters than a "general-purpose" (naïve, "vanilla") fully connected neural network, also called multilayer perceptron (MLP), substantially reducing the number of examples you need to train the network; second, compute power for training large networks increased significantly, mainly through powerful video cards that can efficiently perform matrix-vector multiplications; and third, huge training datasets became available for certain tasks such as image recognition. The first and third points are essential to avoid overfitting of the method.

Unfortunately, many authors appear not to be aware of this, and try to train naïve deep neural networks from thousands of training examples. At first sight, you cannot train a million parameters from thousands of training examples. But machine learning has developed techniques such as regularization (L1, L2, dropout) to avoid overfitting as much as possible: These restrict how freely the parameters can be chosen, so that the effective number of degrees of freedom is much smaller than the number of parameters. In addition, models can be pretrained on large datasets for a related task, then fine-tuned on the few thousand samples (transfer learning).[4] Also, more involved machine learning models such as convolutional or recurrent neural networks reduce the number of free parameters (weights) in the model by sharing weights between positions, to avoid memorizing. Yet, all of these precautions may fail in algorithmic mass spectrometry, due to a peculiarity of the field that it shares with cheminformatics: Our training data is never a random subset of the complete space of samples, and it is very often borderline with regard to being a "trainable subset". Again, consider small molecule research and MS/MS data in reference libraries: Those compounds were not chosen at random but instead, based on availability and price, usually governed by how complicated it is to synthesize the compounds. But if you have a method that synthesizes compound A, then you can also synthesize compound A' which differs from A by, say, a single hydroxy group; and A' will not be much more expensive than A. This implies that for each compound, we may have a cluster of similar compounds in our training data, and this is a major reason why spectral libraries now contain thousands of compounds. In contrast, huge parts of the "chemical space" are still blank spots not only in our training data, but also in all the independent data out there. I conjecture that the improved performance of deep machine learning architectures in cheminformatics and small molecule MS is, at least in part, due to focusing on these peculiarities of the training data; and I have some doubt that these more complex models will perform better than simpler models when evaluated on "novel parts" and blank spots of the molecular structure universe, which will become available in the next decades. Note that the "curse of dimensionality" refers to a related yet deeper problem: In high-dimensional feature spaces, the number of training examples required to cover the space grows exponentially with the number of *features*; even worse, we often have substantially fewer training examples than features. This is no longer a problem of avoiding unsuitable machine learning techniques.

Numerous papers have been published where a naïve deep neural network performs well (and often substantially better than all other machine learning techniques evaluated against) for some dataset which is definitely insufficient for deep learning. Here, the training data not being a random subsample will likely result in a gross overestimation of the method's power; but other problems have been mentioned repeatedly throughout this chapter: Your data are often not independent, allowing machine learning to memorize; you might not learn the problem you are actually interested in but rather a different, much simpler one; you might be training a clever Hans that is "learning" features of the training data you are not interested in. These issues can easily result in a deep neural network showing an outstanding performance.

[4] In a previous version of the textbook, I claimed that you cannot fit millions of parameters using thousands of samples; that is clearly wrong. Did I mention that I am not a machine learning expert? I actively skipped all ML classes while doing my PhD; smart move.

As so often, proteomics is in a much better position than other areas of computational MS: Not only do there often exist huge sets of training data, such as millions of tandem mass spectra of synthetic peptides covering basically the whole human proteome; in addition, proteins and peptides are strings and you can use machine learning techniques developed for text analysis (such as bidirectional long short-term memory networks or, more recently, transformers) mostly "out-of-the-box" [378]. Nevertheless, you should keep your training data structure-disjoint (that is, spectra of the same peptide sequence, for example at different charge states or collision energies, must not appear in different batches, or training and test data) in proteomics, too.

12.10 Filtering vs. priors and the two-step approach

Claude helped!

Throughout this book, we have repeatedly used a two-step approach: First, we generate candidates, potentially far too many; then, we take a detailed look at each of them. For peptide *de novo* sequencing, we compute all peptide strings with close-to-optimal score and re-evaluate them (Sec. 2.8). For glycans, we generate all topologies that fit the data (Sec. 11.3). For molecular formulas, we decompose the monoisotopic mass, then compare the isotope pattern (Chapter 8) or compute a fragmentation tree (Sec. 9.7) for each candidate. And database search is nothing else: The precursor mass selects the candidates, and the fragmentation spectrum decides between them. Let us take a step back and see why this is a good idea, and what we have to be careful about.

The first step is a *filter*, and I mean this in a positive sense: We discard candidates that are in diametral opposition to the data. A molecular formula with a mass 0.1 Da away from the measured mass cannot be the answer, period. Usually, the filter sorts out 99.999999 % of all possibilities, and we lose nothing by doing so. The only requirement is that the filter must be generous, for example regarding the allowed mass deviation, so that the correct answer passes the filter with (almost) certainty.

After filtering, we are left with hundreds or maybe thousands of candidates. Now, two things become possible. First, we can afford to spend much more time on each candidate, and compare it against the data in all detail. Second, now is the time to bring in prior knowledge.

But why not use this prior knowledge right from the start, and save even more time? Because filtering is final. If we use prior knowledge to decide which candidates we generate in the first place, then candidates that look very different from everything we know are gone — even if they would have explained the data perfectly. In Bayesian terms, we have assigned prior probability zero to these candidates, and no amount of data can ever change that: The posterior probability of a hypothesis with prior probability zero is zero. This is known as *Cromwell's rule* [214]. A prior, on the other hand, can be overruled by the data. I do not presume to know the distribution of things which I want to discover in the first place.

For molecular formulas of small molecules, we know the distribution of the ratio between hetero atoms and carbon atoms from molecular structure databases rather well, and we can use it to penalize outlandish candidates (Sec. 8.4). In contrast, the "seven golden rules" of Kind and Fiehn [190] are a filter that is based on prior knowledge; Böcker and Dührkop [28] found that 1.6 % of the compounds in their reference datasets violate these rules. It is a better idea to decompose the monoisotopic mass, compare isotope patterns and fragmentation spectra, and only then look at these rules — as a penalty, not as a filter. This is what SIRIUS does: For about 40 compounds where the correct molecular formula receives a considerable prior penalty, SIRIUS nevertheless identified the correct molecular formula in 25 cases [28].

When searching molecular structure databases with small molecule tandem mass spectra, be warned that a prior that is too strong, is a filter in disguise. Hoffmann *et al.* [153] demonstrate this for "metascores" that combine an *in silico* method with side information from the structure database, such as citation counts: Highly cited "blockbuster metabolites" overshadow all other compounds with the same mass. For example, PubChem contains 8384 compounds with molecular formula $C_{17}H_{21}NO_4$; cocaine has more citations than any of them, whereas more than 8100 of them do not have a single citation and are practically invisible to the metascore. We could remove all invisible compounds from the structure database without ever changing the search results; so, we do not even know which database we are actually searching in. See

also Sec. 12.3 on priors that reward the obvious. Is it ever acceptable to restrict the search space using prior knowledge? Yes, if we notice when the answer is not in there. Hoffmann *et al.* [153] suggest to first search in a smaller, reasonable structure database, and to search PubChem only for those queries that cannot be annotated with high confidence in the smaller database. The prior knowledge (say, "biomolecules are more likely") is used to decide where to look first, but we do not rule out anything.

For peptides and shotgun proteomics, the same question has been discussed: Noble [256] argued that "mass spectrometrists should search only for peptides they care about": To improve statistical power, peptides that are irrelevant for the hypothesis being tested should be removed from the database before searching. Sticker *et al.* [338] replied that "mass spectrometrists should search for all peptides, but assess only the ones they care about" — in short, filter using the data, and apply prior knowledge afterwards.

12.11 Further reading

The literature on biomarker discovery is full of warnings about bias etc.: Back in 1978, Ransohoff and Feinstein [281] warned against bias in the evaluation data and "clever Hans" effects in statistical evaluation. Ransohoff [282] warns against overfitting and improper validation sets in the context of biomarker discovery. Ransohoff [283] remarks that "no guideline can replace an investigator's insight and reflection in considering and addressing possible sources of bias in every step of research, from design and methods to results, analysis and interpretation."

The Tibetan terrier example was brought up by Azulay and Weiss [9], who studied why deep convolutional networks generalize so poorly on small image transformations. The source tag example is from Lapuschkin *et al.* [201], who also popularized the term "clever Hans predictor". Similar warnings for natural language processing were stated by Niven and Kao [255] and McCoy *et al.* [235]. The oldest "clever Hans" warning I am aware of in machine learning, is *more than 30 years old*: In 1992, Dreyfus and Dreyfus [86] told the "legend" that a neural network did not detect tanks in pictures, but rather differentiated between pictures taken on sunny and cloudy days. Next, see the technology feature by Marx [232] in *Nature Methods* on practical pitfalls, including the "curse of dimensionality". Riley [291] from Google recently wrote a comment about pitfalls in machine learning: Namely, inappropriate splitting of the training data (Sec. 12.5), clever Hans predictors (Sec. 12.7), and wrong objectives when learning. These warnings are not new but since it is *from Google* and published *in Nature*, there is a chance that people will finally understand this is an issue. Also see the numerous articles in the *Quanta Magazine* (`https://www.quantamagazine.org/`) on the same issue.

With regard to feature selection, Quinn [276] found that out of 102 articles where machine learning models were trained for the analysis of microbiome (usually, stool) samples, only about 20 % set aside a test set at all, and only 14 studies avoided leaking the test set during feature selection. In the end, only 12 % of the studies reported performance metrics that can be trusted. These are shockingly small percentages.

Mayer *et al.* [234] show how you can draw an elephant with four (complex) parameters, and how you can make him wiggle his trunk with the fifth. Chari and Pachter [53] take this to the next level: They show that an embedding of single-cell data forced into the shape of an elephant preserves the structure of the data about as well as the well-known Uniform Manifold Approximation and Projection (UMAP) method.

See Gatto *et al.* [120] for general "do's and don'ts" of computational methods evaluation. I have read and reviewed many papers where people did not adhere to the advice given in this section, but this is not "blame and shame". When reading a paper, just decide for yourself.

The Master of the Obvious is a recurring character from the Dilbert comics.

Filtering has a long history in sequence analysis: BLAST [4] aligns only those database sequences in detail that share a short, high-scoring word (a *seed*) with the query. A filter is called *lossless* if it provably never discards a true hit; the q-gram lemma is the classical example [48, 165]. BLAST is not lossless, and its seeds have led to a theory of how to choose seeds so that we lose as few true hits as possible, starting with the spaced seeds of PatternHunter [177, 225].

Claude helped!

12.12 Exercises

12.1 Consider the HIV rapid test. There are two models, M_1 for HIV infection and M_0 for no infection; similarly, $D \in \{\text{pos}, \text{neg}\}$ indicates if the test was positive for an HIV infection or not. Assume that the rapid test has specificity and sensitivity 99 %. What is $\mathbb{P}(\text{pos} \mid M_1)$, $\mathbb{P}(\text{pos} \mid M_0)$, $\mathbb{P}(\text{neg} \mid M_1)$, and $\mathbb{P}(\text{neg} \mid M_0)$? If only 0.01 % of the tested individuals have an HIV infection, what are the prior probabilities $\mathbb{P}(M_0)$ and $\mathbb{P}(M_1)$? Finally, calculate the posterior probability $\mathbb{P}(M_1 \mid \text{pos})$ of an HIV infection if the test is positive.

13 The missing chapters

> *"The great thing about the Internet is that everyone can finally give their opinion to the whole world. The terrible thing is that everybody does it."* (Marc-Uwe Kling, *Die Känguru-Chroniken*)

UNFORTUNATELY, this book is vastly incomplete. Despite my deliberate limitation to topics that are *interesting on an algorithmic level*, I nevertheless feel that certain aspects should have been covered. In fact, I have stuck to things that I can rather safely talk about. In case you want to contribute to this textbook, please contact me! Below are a few topics that should be covered, or justifications for other topics which I believe I cannot cover here:

- **Charge states.** If there is an isotope pattern, then the charge state can easily be deduced from it; there are very few situations where this is not the case. Nevertheless, some words on how the determination of charge state can be carried out in a statistically robust way, would be helpful. If there is no isotope pattern, as for a fragmentation spectrum from the monoisotopic peak, then you probably have to keep all possible interpretations as alternatives.

- **Signal processing and peak picking.** This is a very different topic from the questions covered so far in this textbook. To give the reader an (imprecise) impression, I have covered a few details of this step below (Sec. 13.1).

- **Deisotoping.** Experimental spectra contain isotope peaks, which are often considered an annoyance. Hence, removing these isotope peaks from an MS1 or MS/MS spectrum has been considered repeatedly in the literature. In many cases, the better solution would have been to use the isotope pattern to your advantage, as we have seen that it contains much information. But for certain experimental setups such as isotope labeling or imperfect mass filters, it is indeed challenging to identify the monoisotopic mass of a molecule or to estimate the corresponding isotope pattern.

- **Aligning single-stage mass spectra.** Unless you label all molecules in one of your samples (chemically or metabolically), you have to compare two samples as two individual MS1 runs. Unfortunately, that means that all peaks will be slightly shifted; in particular, the retention time can vary between two measurements. Establishing a mapping between two LC-MS runs is further complicated by the fact that some peaks are present in only one of the two runs. Algorithmically, this boils down to alignments, time-warping or nonlinear regression.

- **Protein inference.** We have seen how to sequence peptides and how to search for them in a database, plus estimating the significance of such hits; but what we really want to know, is what proteins (not peptides) are present in the sample. One of the oldest, probably the best-known and possibly still the most-used approach for this is ProteinProphet by Nesvizhskii, Keller, Kolker and Aebersold [249].

- **Protein shotgun sequencing.** This problem, not to be confused with shotgun proteomics, is algorithmically very challenging and gives rise to several interesting combinatorial problems [11].

- **Tag-based approaches.** You first use *de novo* sequencing to derive a short "tag" (3–6 amino acids) of the peptide, then restrict your database search to those peptides that have the tag. This is all about speeding up database search, and "speeding up things" is definitely in the realm of algorithmics. Doing so is beyond a nice-to-have feature: Using a tag-based approach, you could do database

search with practically unrestricted post-translational modifications! (Certain restrictions apply.) Tag-based searching has been revived for open modification search, but fragment-ion indexing has become the more popular route. Potentially, one reason is the advent of Data-Independent Acquisition (DIA, still funny) where a fragmentation spectrum contains peaks from 10+ peptides. This results in the challenging task that the fragmentation spectrum contains k peptides, and you have to sequence short tags for each of them — give it a try. Also, tag-based searching may interfere with false discovery rate estimation. See Ma and Johnson [224] and Muth, Hartkopf, Vaudel and Renard [247] for reviews.

- **Data-Independent Acquisition (DIA).** In DIA, chimeric fragmentation spectra are the rule rather than the exception, see Sec. 4.11; this requires entirely different algorithms. Most tools follow the *peptide-centric* approach introduced with SWATH [125, 347]: For each precursor ion in a spectral library, we extract the chromatograms of its fragment ions from the DIA data, and search for an elution peak where these chromatograms co-elute. Candidate peaks are scored by co-elution, mass accuracy, and similarity to the library spectrum; these scores are combined into a single score, and decoy precursors are used to estimate false discovery rates, see Chapter 6. The spectral library is either measured, or predicted from a protein sequence database (*library-free* mode). The most widely used tools today are the commercial Spectronaut [46] and DIA-NN [80]; DIA-NN combines its scores using an ensemble of neural networks, and resolves interference if several precursors are matched to the same elution peak. In contrast, *spectrum-centric* approaches search the DIA spectra themselves, either after building pseudo-spectra from co-eluting precursor and fragment peaks [350], or directly using fragment-ion indexing [375]. See Lou and Shui [218] for a review, and Lou *et al.* [217] for a benchmark of commonly used tools.

Claude helped!

- **Simulating fragmentation spectra.** For peptides, Zhou, Zeng, Chi, Luo, Liu, Zhan, He, and Zhang [378] show how to predict fragmentation spectra using deep learning; these spectra are of excellent quality and "look like the real thing". Unfortunately, the method description would have to be either very short and shallow, or extremely intricate — if I explain bidirectional Long Short-Term Memory (LSTM) neural networks first. The complete methods description in [378] is only 44 lines; citing Zhou *et al.* [378], the authors are "looking forward to more nontrivial applications of deep learning in proteomic studies". Both options are somewhat unsatisfactory. For metabolites, quantum chemistry simulations are definitely beyond the scope of this textbook, and the Competitive Fragmentation Modeling method of Allen *et al.* [1, 2] is an intricate and non-trivial blend of machine learning and expectation maximization, plus a hint of Markov chains. Recently, papers have been published that claim to solve the problem via machine learning; but it looks to me as if these papers have major evaluation issues, see Chapter 12 and in particular Sec. 12.8.

13.1 Signal processing and peak picking

Computational mass spectrometry applications often have the form of a pipeline, where each stage can access the information computed on previous stages. The first stage of this pipeline is processing the raw MS data, enhancing the signals and suppressing the "noise". Even though sensitivity, signal-to-noise ratio, and mass accuracy have improved considerably with modern instruments, the automated analysis of raw MS data is still a delicate task, complicated by low-abundant molecules, chemical and electronic noise, and overlapping patterns.

MS instrument manufacturers tend to sell their instruments with proprietary signal processing software. As always, using such proprietary software comes with its pros and cons: The software is often specifically tailored for a particular MS instrument, so there is no need for lengthy parameter optimization; it usually ships with a decent user interface; and, it reads and writes the manufacturer's proprietary file formats for the raw data. Most importantly, the vendor signal processing software is often deeply entangled with the software that operates the MS instrument. On the "cons" side, the methods and algorithms of the

software are usually not documented; and, the techniques and algorithms used within such software are rarely as evolved and sophisticated as the ones found in the academic literature.

We have assumed throughout this textbook that mass spectra are presented to us in the form of a peak list, consisting of masses, intensities, and possibly further features of each detected peak. The *classical* way to derive such peak lists consists of four steps:

1. First, the baseline is removed from the spectrum: In particular for MALDI-TOF spectra, this is a systematic offset caused mostly by matrix clusters and chemical noise. The baseline is slowly varying and smooth, and usually diminishes with higher *m/z* values. Standard techniques for removing the baseline include morphological filtering and sliding window approaches.

2. Then, noise is removed from the spectrum. We first have to understand that in mass spectrometry, "noise" may refer to two very different kinds of artifacts. A random noise component is present in all experimental datasets, consisting of several components such as high-frequency jiggle and "shot noise". This noise can be removed by techniques such as sliding averages (in particular, Gaussian smoothing), Savitzky-Golay filters, or the LOWESS (locally weighted scatterplot smoothing) transform. On the other hand, certain signals were indeed generated by ions, but not by molecules from the sample (matrix or solvent clusters, contaminants), and these are called "chemical noise". Usually, signals from chemical noise cannot be filtered at this step, unless we have a quite good understanding of what we are searching for.

3. Next, peaks have to be detected; this is referred to as "peak picking" or "centroiding". (The word "centroiding" is a misused scientific term,[1] because centroiding is at best one of several steps of the peak detection process; but it is definitely too late to get rid of it.) In principle, peak detection can be performed by searching for local maxima; but as not each spike in the mass spectrum will correspond to a real mass peak, peak detection algorithms employ techniques such as sliding averages of first and second derivative (or much more involved math) to improve their robustness.

4. Finally, we can fit a template function, describing the expected peak shape such as Gaussian or Lorentzian, to those regions of the signal where peaks were detected. This will get rid of peak shapes that clearly indicate a "noise peak". Non-symmetric peak shapes such as "exponentially modified Gaussian" have also been exploited for peak fitting. Clearly, peak fitting may require substantial additional running time.

All detected (and fitted) peaks are then evaluated, in the simplest case by their intensity, and peaks above some threshold are reported in a peak list.

Starting in the second half of the 2000s, there appears to be a paradigm shift regarding MS peak picking: Methods published since then often utilize wavelets for this task [49, 88, 244, 252, 326]. Wavelets are small, wave-like functions, and allow us to split the mass spectrum simultaneously in the mass and the frequency domain. The wavelet transform is similar to the well-known Fourier transform, but the wavelet transform allows us to split the mass spectrum locally, which is required for peak picking. Since true signals, high-frequency noise, and low-frequency baseline are located in different frequency domains, there is no need for baseline correction or noise filtering; instead, the wavelet transform is applied directly to the raw MS data. For today's high-resolution instruments, the more demanding task is detecting features in the retention-time dimension; here, too, wavelets are frequently used, and more recently machine learning.

In case the MS instrument is coupled to LC (liquid chromatography) or GC (gas chromatography), peak picking for MS1 can either be done in one dimension, processing mass spectra individually, or in two dimensions (mass and retention time). Two-dimensional approaches are usually more robust, as the differentiation between signal and noise becomes easier if we look at both dimensions simultaneously.

[1] What is the technical term for a misused scientific or technical term? I thought it was "malapropism" but this turns out to be a misused technical term…

Early software returned only tens of peaks for each spectrum. This was clearly tailored toward a human who has to look through these lists. But for many years, these peak lists have in most cases only been the input of the next computational analysis step, as those mentioned in this textbook. If you are developing automated methods that take peak lists as input, then never let the peak picking software throw away peaks for you! Rather, ask the peak picking software to output hundreds or even thousands of peaks, *including* intensities or "significances" or whatever criterion, and decide yourself which of them are significant for your method and which are not. Even these hundreds of peaks are signals selected by the instrument software and the peak picking algorithm; they are not the physical truth, and there is no such thing here. This may require some additional computing time for peak picking — throwing them away comes for free. But it mostly avoids those nasty cases where your algorithm returns a wrong answer, just because several tiny but clearly visible peaks that could have saved the day, were discarded by the peak picking algorithm.

14 Gazing into the Crystal Ball

> *"Maybe you'll marry, maybe you won't, maybe you'll have children, maybe you won't, maybe you'll divorce at 40, maybe you'll dance the funky chicken on your 75th wedding anniversary."* (Mary T. Schmich, *Advice, like youth, probably just wasted on the young*)
>
> *"Prediction is very difficult, especially about the future."* (Karl Valentin, Mark Twain, Niels Bohr, Robert Storm Petersen, or Karl Kristian Steincke)

As said in the preface, I have tried to focus this textbook on questions which I assume to pop up regularly when dealing with mass spectrometry data; I have done so because the actual scientific questions we approach, may change rather drastically over a period of ten years. But where is the journey going? Quite honestly, I do not have a clue; "my advice has no basis more reliable than my own meandering experience."

Is mass spectrometry here to stay? As a bioinformatician, you might be anxious to invest time and learn about computational mass spectrometry, given that you can hardly use any of the computational methods from this book for, say, the analysis of next-generation sequencing data. Does it make sense to think so much about computational mass spectrometry, when a new technology might be just around the corner?

Generally speaking, new technologies usually do not completely replace older technologies; rather, the older technology remains in use, sometimes only in certain niches: For example, there are more than 1500 coachmen working in Germany, more than 100 years after the invention of the automobile. Deciding on the "best" technology is usually not a one-dimensional thing. The same is true in science: Whereas microarrays have stopped being the "next big thing" in life science more than a decade ago, they are in heavy use in clinical settings. Even if new technologies will come along for the analysis of proteins, metabolites etc., it is unlikely that these new technologies are better than mass spectrometry *in every possible aspect*. But this is probably just a weak consolation.

Where I was wrong. I have to admit that a few years ago, I might have given you different advice, suggestions and solutions for some of the problems and questions raised in this book. On the one hand, this is a classical "if only I had known then what I know now" situation. On the other hand, science is a moving target, and "it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!" To me, that is the beauty of it.

Here is an incomplete list of questions where I was wrong and had to change my mind, over the years. I include this to warn you that there are no perpetual truths outside of math. Also, it may inspire you to try out things that you do not believe in, and to think "outside the box".

- I never liked the scalar product of functions to compare mass spectra (Sec. 4.4). Then, Heinonen *et al.* [149] successfully used the probability product kernel for machine learning applied to small molecules. Nice surprise.
- When scoring intensity deviations, I was an advocate of the relative error model h'/h over the absolute error model $h'-h$ (Sec. 4.8). Turns out that the latter performs slightly better in practice. But what I did not expect is that the combination of absolute and relative error performs substantially better in practice than the two individually.
- I believed that modeling noise peak intensities using an exponential distribution (i.e., simply use the intensity as part of your score) will do the trick — that is, you probably find a better-suited

distribution but the improvement will be negligible in practice. Kai Dührkop proved my intuition wrong [28].

- I did not think that deep learning could have much impact in mass spectrometry as it requires millions of training examples. Until labs started to measure hundreds of thousands of synthetic peptides, resulting in millions of training spectra.

Please, prove me wrong, too! And tell me what I have to correct.

15 Mathematics and computer science

"Do not worry about your difficulties in Mathematics. I can assure you mine are still greater." (Albert Einstein)

"To infinity and beyond!" (Buzz Lightyear)

Whereas this textbook is mostly targeted at an audience familiar with mathematics and computer science, there may be readers who are not. To this end, I provide a very short and crude introduction to some of the relevant topics touched upon in this textbook.

15.1 Writing numbers

It must be understood that when writing a real-valued number $x \in \mathbb{R}$ in this or any book, we cannot write down all its digits: For example, e and π would require us to write an infinite series of digits, and the same is true for every irrational number such as $\sqrt{2}$. So, when we write $\pi = 3.1415926535$ for a real-valued number, this does not mean that the sequence of digits stops at this point. But this is the case for practically *every real-valued number*, so we do not have to use a special notation for this fact. (Technically speaking, the rational numbers form a null set, and the set of numbers with terminating decimal notation is a subset of the rational numbers. In even more detail, the rational numbers are a countable set — this is a classical exercise for first-year math students — and every countable set has Lebesgue measure zero. Even uncountable sets can have Lebesgue measure zero; I can still remember the exercise, "show that the Cantor set is not countable and has Lebesgue measure zero" from the second year of my math studies.) Depending on what we have in mind with the number, we can round it — but this should be noted — or, we simply stop at any point of the digit series: So, $\pi = 3.1415926535$ simply tells us nothing about the next digit of π; it turns out to be 8. On rare occasions, I will add dots to the end of the decimal notation, to remind the reader that this is not all.

We use the scientific notation $1.23 \cdot 10^{12}$ or $1.23 \cdot 10^{-12}$ for very large or very small numbers. The coefficient x in $x \cdot 10^y$ must satisfy $x \geq 1$ and $x < 10$, and is usually given with at least two decimals. Please, do not write "$x * 10^y$" as the operator '$*$' is reserved for a different mathematical operation (convolutions); if you have to, you can write "$x \times 10^y$".

15.2 Masses

Throughout this textbook, we assume that masses are integers or real-valued. Some care has to be taken: For integers, we only have to make sure the computer *knows* this is an integer. If you do not tell the computer that some mass is an integer, it might use a float instead, and this can lead to tremendously funny debugging sessions later.

The situation is much more complicated for real-valued masses. A few things are essential, though:

- Use *double* precision, not single precision. When computers deal with real-valued numbers, they can do this only with some fixed precision, see below. There are very few cases where intentionally sacrificing accuracy in your computations makes sense.[1] Otherwise, always use the full precision

[1] Computer graphics computations are one example; training neural networks is a second, where using less accuracy allows for faster training of larger models. Finally, saving memory in tremendously memory-intense computations is a third example: But there, intermediate computations should be carried out with double precision, and care has to be taken.

your computer can offer you. This usually comes at little or no cost with respect to running time: For scalar computations, modern processors multiply double precision numbers about as fast as single precision numbers. (Certain restrictions apply: Vectorized code on CPUs, and in particular graphics cards, can multiply *more* single precision numbers per time step.)

- For predicting spectra and theoretical masses, use numbers with the best accuracy you can get. The masses in Tables 1.1, 2.1, 7.1, and 7.5 are not meant for the computer, but rather for the human reader. Instead, you should download masses with higher mass accuracy from the Internet.[2] Masses in these tables are given with only six digits behind the decimal point. For H, this implies a relative mass error of up to 0.5 ppm. Relative mass accuracy for the other elements is better. Still, you add a mass error of, say, up to 0.1 ppm to your data, which is completely independent of the instrument and can easily be evaded.

15.3 Floating-point arithmetic and numerical stability

Computers implement real numbers as floating-point numbers $s \cdot 2^e$ where s is the *significand* and e is the *exponent*, such that $1 \leq s < 2$. Both numbers are stored only with a limited precision: For *single precision*, we have $-126 \leq e \leq 127$ and 23 bits for the significand not including the implicit first bit. So, the smallest difference between two significands is $2^{-23} = 1.192 \cdots \cdot 10^{-7}$ or, put differently, only slightly more than 0.1 ppm. Whereas the restriction of the exponent is not a problem for computational mass spectrometry — we can represent numbers as large as $3.4 \cdot 10^{38}$ — restriction of the significand is. In comparison, *double precision* uses 52 bits for the significand (again not including the implicit first bit) and 11 bits for the exponent, so $-1022 \leq e \leq +1023$.

As we are using floating point arithmetic, we have to keep in mind that statements that are true for real numbers, are not necessarily true for floating point numbers. For example, $0.1 \cdot 0.1 - 0.01$ does not equal zero when using (binary) floating point arithmetic. Using single precision, 0.1 is encoded in the computer as 0.100000001490..., and $0.1 \cdot 0.1$ is calculated as 0.010000000707.... In contrast, 0.01 is encoded as 0.009999999776..., so computing $0.1 \cdot 0.1 - 0.01$ with single precision will numerically result in $0.0000000009313 = 9.313 \cdot 10^{-10}$. When subtracting numbers, this can lead to unexpected behavior, in particular when the exact result is close to zero. As the operands of a comparison usually carry such rounding errors from previous computations, it should be understood that comparing two floating point numbers for equality is usually also a bad idea. When we do a series of arithmetic operations, errors can get "large" — large in comparison to the machine precision, that is. Involved algorithms have been developed to keep rounding error accumulation to a minimum, such as the Kahan summation algorithm. For us, it is usually sufficient to keep in mind that we should never use single precision arithmetic, as rounding error accumulation can easily rise to a level beyond the mass accuracy of an MS instrument; and not to compare two masses for equality but instead, allow for a small $\varepsilon > 0$ even if both masses are derived from theoretical computations.

If you want to learn more about how computers deal with real-valued numbers, take a look at Overton [261].

15.4 Graph theory

Graphs are everywhere in this book: spectrum graphs for peptide *de novo* sequencing (Chapter 2), fragmentation graphs and fragmentation trees (Chapter 9), molecular graphs (Chapter 10), and glycan trees (Chapter 11). Graph theory goes back to Leonhard Euler who, in 1736, showed that there is no walk through the city of Königsberg that crosses each of its seven bridges exactly once. Fittingly for this book, the word "graph" was coined by James Joseph Sylvester in 1878, inspired by the diagrams chemists used to draw molecules. In the following, I only introduce what we need; see Diestel [81] for a proper introduction.

Claude helped!

[2] https://www-nds.iaea.org/amdc/ame2020/mass_1.mas20.txt

A *graph* $G = (V, E)$ consists of a finite set of *nodes* (or vertices) V and a set of *edges* E. For an *undirected* graph, an edge connects two nodes u, v without any direction; for a *directed* graph, an edge $(u, v) \in V \times V$ points from u to v. In both cases, we write uv for short. Usually, we use m, n to denote the size of a graph, where $n := |V|$ is the number of nodes and $m := |E|$ the number of edges. In an undirected graph, the *degree* of a node is the number of edges incident to it; in a directed graph, we distinguish the *in-degree* (number of incoming edges) and the *out-degree* (number of outgoing edges).

Molecular graphs can be modeled as *multigraphs*, where two nodes can be connected by more than one edge, see Sec. 10.2. Alternatively, we can think of edges having a *label* $\ell : E \to N$ for some reasonable set N, say, $N = \{\text{single, double, aromatic, } \dots\}$. Nodes may carry labels or *colors* $c : V \to C$, such as the element of an atom, and edges may carry *weights* $w : E \to \mathbb{R}$.

Paths, cycles, and directed acyclic graphs (DAGs) are defined in Sec. 2.3; paths in undirected graphs in Sec. 10.2. In short: A *path* $p = u_0 u_1 \dots u_l$ is a sequence of nodes such that $u_{i-1}u_i$ is an edge for all $i = 1, \dots, l$, and $|p| := l$ is its *length*; in a directed graph, the edges must point from u_{i-1} to u_i. Nodes may repeat. In a directed graph, a *cycle* is a non-trivial path from a node to itself, and a directed graph without cycles is a *directed acyclic graph* (DAG). A *cycle* in an undirected graph is a non-trivial path from a node to itself that uses each edge at most once. An undirected graph is *connected* if there is a path between any two of its nodes; the maximal connected subgraphs of a graph are its *connected components*. The nodes of a DAG can always be sorted in *topological order*, such that every edge points from an earlier to a later node; this order, or its reverse, is the order in which dynamic programming visits the nodes (Sec. 15.5). For spectrum graphs, sorting nodes by increasing mass gives a topological order; for fragmentation graphs, sorting by decreasing mass.

A *tree* is a connected undirected graph without cycles. A tree with n nodes has exactly $n-1$ edges, and adding any edge creates a cycle; this is the basis of the RDBE value in Sec. 8.4. A *rooted tree* (or *arborescence*) is a directed graph with a distinguished root node, such that the root has in-degree zero, all other nodes have in-degree one, and every node can be reached from the root. Nodes with out-degree zero are called *leaves*. Fragmentation trees and glycan trees are rooted trees.

A *subgraph* $G' = (V', E')$ of $G = (V, E)$ satisfies $V' \subseteq V$ and $E' \subseteq E$. It is the *induced* subgraph of V' if E' contains *all* edges of E between nodes of V'. In our model, fragments of a molecule correspond to connected induced subgraphs of the molecular graph, whereas a fragmentation tree is a subgraph of the fragmentation graph that is usually not induced.

A graph is *bipartite* if its nodes can be split into two sets such that every edge connects nodes from different sets; matching the peaks of two mass spectra (Chapter 4) results in such a graph. A graph is *planar* if it can be drawn in the plane without crossing edges; by Kuratowski's theorem, this is the case if and only if it does not contain a subdivision of K_5 (five nodes, all connected) or $K_{3,3}$ (three nodes connected to three other nodes) as a subgraph (equivalently, as a minor). Two graphs are *isomorphic* if they become identical after renaming the nodes; counting molecular structures or glycan topologies means counting graphs *up to isomorphism*, which is what makes it hard.

Many problems on graphs have efficient algorithms, such as finding shortest paths in a graph with non-negative edge lengths, or longest paths in a DAG [65]. Others are NP-hard, such as finding a longest path in a general graph where no node may be used twice, or a maximum colorful subtree; see Sec. 15.8. Questions connected to isomorphism turned out to be particularly spicy: For example, checking whether one graph is a subgraph of another, modulo isomorphism, is an NP-hard problem; checking whether two graphs are "basically identical" (isomorphic) is one of the few natural problems in NP for which we know neither a polynomial-time algorithm nor a proof of NP-completeness.

15.5 Dynamic programming

> *"What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, 'programming.' I wanted to get across the idea that this was dynamic,*

> *this was multistage, this was time-varying [...] Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."* (Richard Bellman, *Eye of the Hurricane*)

Dynamic programming computes the solution to a given optimization problem by combining optimal solutions to certain subproblems. In addition, the space of subproblems we have to solve is sufficiently small. Consider Fibonacci numbers (this is not an optimization problem but shows the trick): The equality $F_i = F_{i-1} + F_{i-2}$ can be directly translated into a recursive function. But this means that we have to compute the same values over and over again, resulting in a terribly slow algorithm. Instead, we use an array $F[\cdot]$ to store the Fibonacci numbers, initialize $F[1] \leftarrow 1$ and $F[2] \leftarrow 1$, and finally compute $F[i] \leftarrow F[i-1] + F[i-2]$ for $i = 3, \ldots, n$ until we reach the number $F_n = F[n]$ we are interested in. For Fibonacci numbers, there exists a closed formula, see (3.4) in Sec. 3.2; but that is usually not (almost never) the case for dynamic programming.

Dynamic programming can either be formulated "bottom-up" or "top-down". Here, *top-down* means describing how an optimal solution of an instance can be computed from optimal solutions of smaller instances, as in the many recurrences from Chapter 3, see for instance (3.2) on page 43. Usually, this form is easiest to understand for a human reader. In contrast, *bottom-up* means that the optimal solutions of smaller instances point to all larger instances where they might contribute, and update the scores of these instances accordingly. An example of an algorithm that is taught bottom-up in computer science classes, is Dijkstra's algorithm for finding a shortest path in a graph.

But when it comes to implementation, bottom-up dynamic programming often has the upper hand. This can be the case when the order in which elements of some set have to be processed, is not immediately clear from the problem formulation. Not all DP problems have such a crystal clear structure as pairwise sequence alignment or the problems from Chapter 3, where it is obvious how to fill the tables.

Consider again Dijkstra's algorithm: We are given a directed graph $G = (V, E)$, and edge lengths $l(u, v) \geq 0$ for every edge (u, v) of G. We search for the shortest paths from some fixed root node to all other nodes of the graph. We use an array $D[\cdot]$ that, when the algorithm terminates, contains exactly this length. One can formulate Dijkstra's algorithm top-down, saying that $D[v]$ is the minimum of $D[u] + l(u, v)$ over all incoming edges (u, v). But now, nodes in the graph have to be processed in an order so that $D[u]$ for all predecessors u of v has been computed before we actually compute $D[v]$. The bottom-up formulation of the algorithm intrinsically solves this problem, as only entries that cannot be updated themselves, are used to update entries in the table. This works because all edge lengths are nonnegative; for graphs with cycles, an order as required by the top-down formulation does not even exist.

I mentioned above that computing Fibonacci numbers is not an optimization problem; hence, this is not dynamic programming but rather a *recurrence*, where one value is computed using others previously computed and stored. At least in bioinformatics, it has become common practice to also refer to such recurrences as "dynamic programming", see the many papers at bioinformatics conferences that carry the term DP in their title. Being a bioinformatician, I have been "scientifically raised" with this impreciseness. Consequently, throughout this textbook, I may also speak about "dynamic programming" when I should rather say "recurrence".

Usually, we are not so much interested in the score of the optimal solution, but rather in its structure. Throughout this textbook, the term "backtracing" will refer to "computing a single optimal solution", whereas "backtracking" (notice the 'k') refers to "computing all optimal or even suboptimal solutions". See Chapter 2 (for example, Sec. 2.4) and Chapter 11 (for example, Exercise 11.5) for more details on how this is done.

15.6 Logarithms

Throughout this textbook, you will often find the notation "$\log x$", indicating that you should calculate the logarithm of x. There are two different logarithms that you get to learn at school, namely $\log_{10} x$ as

the logarithm with base 10, and $\ln x$ as the natural logarithm with base e, where $e = 2.7182818284$ is Euler's number. The base 10 makes it easy to interpret results in the decimal system, whereas the base e is preferred by mathematicians, as it has a particularly nice property: If $f(x) = e^x$ then $f' = f$. In addition, many computer scientists (and certain branches of mathematics) use the binary logarithm $\mathrm{lb}\, x$ with base 2. Now, which logarithm is the correct one?

In fact, the situation is more complicated and, at the same time, much simpler: First, note that we have an infinite number of bases $b > 1$ to choose from, and for each such base, there is a logarithm function $\log_b x$. In particular, we have $\ln = \log_e$ and $\mathrm{lb} = \log_2$. Now, if you find the expression "$\log x$" in this textbook or in the literature on complexity theory and algorithmics, this means that you can take an *arbitrary* base to carry out your calculations, as long as the base is fixed: That is, you carry out all calculations with the same base. There are few cases where a particular base is required; but this will be explicitly denoted.

Why can we be so careless about the base of the logarithm? Because

$$\log_b(x) = \frac{1}{\log b} \log(x)$$

where "log" is the logarithm to an arbitrary base. Hence, changing the base only introduces a constant factor. This factor vanishes anyway in the big-O notation.

15.7 Approximations

There exist at least three meanings of the word "approximation", plus the informal use:

1. *Approximation theory* is a field of mathematics that studies how functions can be approximated with simpler functions. An example is the approximation of a function using polynomials, in particular Chebyshev approximation. Approximation theory makes it possible to quickly evaluate the exponential or logarithm function on your computer. This is not what I am talking about throughout this book.

2. In computer science, *approximation algorithms* are algorithms that find a solution to an NP-hard optimization problem with provable guarantees. These algorithms usually have polynomial running time. Unless P = NP, we cannot find exact solutions to NP-hard optimization problems in polynomial time; therefore, it is a natural question to ask "how close we can come" in polynomial time. The guarantee of an approximation algorithm can be a constant (for example, "the solution has at most two times the cost of an optimal solution") or dependent on parameters of the instance (for example, a factor of $\log n$). Polynomial-Time Approximation Schemes (PTAS) allow approximation with "arbitrary quality" at the cost of increasing running times. Approximation algorithms are, for the vast majority of problems, not of much use in bioinformatics; bioinformaticians are usually interested in the *structure* of the optimal solution, but not the value of the objective function [91]. There are a few places throughout this book where I speak about approximation algorithms.

3. In most of the cases, when I speak about an approximation, the correct mathematical term would be "*asymptotics*": For two functions $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) \sim g(n)$ if and only if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

That is, the relative error vanishes if you see one of the functions as an approximation of the other. This is a precise mathematical notation, and we have to prove that it holds. Unfortunately, asymptotics does not tell us anything about the rate of convergence; to reach a reasonable relative error, we might have to consider values of n beyond a googolplex, $n \geq 10^{(10^{100})}$. (A googolplex is a number which is much larger than anything you ever have to consider in real life and most of science, but still puny in comparison to infinity.) Furthermore, the absolute error can become arbitrarily large

with increasing n. To make the asymptotic function a useful approximation in the colloquial sense of the word, we have to check whether $f(n)$ is close to $g(n)$ for reasonably small n, too.

4. Colloquially speaking, "f approximates g" means that "f is somewhat close to g". This does not come with any guarantees about how close f is to g. I will try to clearly indicate if I use "approximation" with this meaning.

15.8 Computational complexity

Claude helped!

How fast is an algorithm? Measuring seconds depends on the computer, the programming language, and the person who implemented it. Instead, we count the number of elementary operations (additions, comparisons, memory accesses) as a function of the input size n. Throughout this book, we assume that each such operation takes constant time, independent of the numbers involved. This *unit-cost* model is unrealistic for huge numbers: In cryptography, numbers with thousands of digits are the daily bread, and the time for multiplying two such numbers grows with their length. For our applications, it is perfectly fine: Masses, intensities, and scores easily fit into a double precision floating-point number or a 64-bit integer, see Sec. 15.3. Should intermediate numbers ever become astronomically large (say, beyond 10^{100}), we will not need them exactly anyway. This is different in cryptography, where every bit counts.

We usually do not care about constant factors, as these depend on the implementation, too. For functions $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, we write $f(n) = O(g(n))$ if there exist constants $c > 0$ and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$; so, f grows at most as fast as g. Analogously, $f(n) = \Omega(g(n))$ means that f grows at least as fast as g, and $f(n) = \Theta(g(n))$ means both. For example, $3n^2 + 100n \log n = \Theta(n^2)$. An algorithm has *polynomial running time* if its running time is $O(n^c)$ for some constant c, that is, polynomial in the input size. In contrast, a running time such as $\Theta(2^n)$ is *exponential*: Adding a single element to the input doubles the running time.

Be careful what the "input size" is: An integer M is encoded using about $\log_2 M$ bits. An algorithm with running time $O(kM)$ for an alphabet of k masses, such as the dynamic programming algorithms for decomposing masses in Chapter 3, is therefore *not* polynomial in the input size, but exponential. Such algorithms are called *pseudo-polynomial*: They are polynomial if the numbers in the input are small, which is often the case in mass spectrometry.

Computational complexity theory sorts problems, not algorithms. It is usually formulated for *decision problems* with answer "yes" or "no". If we are interested in an optimization problem, we ask "is there a solution with score at least t?", and voilà, we again have a decision problem. Problems are sorted into classes: For example, the class P contains all decision problems that can be solved in polynomial time. The next class is slightly more complicated: Class *NP* contains all decision problems where a "yes" answer can be *verified* in polynomial time, given a suitable certificate. To explain what a certificate is, is beyond the scope of this textbook, so a short example will have to do: If someone shows us a molecular formula and claims that this is a decomposition of a certain mass, we can quickly check whether the masses add up. Clearly, P $\subseteq$ NP: If you can solve the problem in polynomial time, you can simply ignore the certificate.

A problem is *NP-hard* if every problem in NP can be reduced to it in polynomial time; so, a polynomial-time algorithm for one NP-hard problem would give us polynomial-time algorithms for all problems in NP. An NP-hard problem that is itself in NP is *NP-complete*. Cook [62] showed that the satisfiability problem of Boolean formulas is NP-complete, and Karp [175] showed NP-completeness for 21 problems; Garey and Johnson [118] is the classical catalogue. To show that a new problem is NP-hard, we *reduce* a known NP-hard problem to it. Whether P = NP is the most famous open question of computer science, and one of the seven Millennium Prize Problems. Most researchers believe that P $\neq$ NP; then, there is no polynomial-time algorithm for any NP-hard problem. Others believe that P = NP might hold, but that it will not have any practical consequences: The existence of a polynomial-time algorithm does not mean that the algorithm is efficient in practice. For example, an algorithm with running time $O(n^{10000})$ is polynomial-time but cannot be applied for any reasonable n.

We indicated above that problems with numbers in the input require extra care. The MONEY CHANGING problem from Chapter 3 is NP-hard, but can be solved in pseudo-polynomial time; such problems are *weakly* NP-hard. A problem is *strongly* NP-hard if it remains NP-hard even if all numbers in the input are bounded by a polynomial in the input size.

NP-hardness tells us that we should not expect an algorithm that is both exact and fast for *all* instances; it does not tell us that a problem is hopeless in practice. There are several ways out: Exponential-time exact algorithms may be fast enough for the instances we care about; *fixed-parameter algorithms* confine the exponential part of the running time to a parameter that is small in practice (Sec. 9.4.1), see Downey and Fellows [85] and Niedermeier [253]; integer linear programming solvers often work surprisingly well (Sec. 9.4.2); approximation algorithms come with a guarantee on the score (Sec. 15.7); and heuristics come with no guarantee at all, but may be blindingly fast. See Kleinberg and Tardos [192] for a readable introduction to algorithms and computational complexity.

16 Conclusion

> *"It is not surprising that widely used packages [in computational MS] are from large MS labs [...]. The music in proteomics is in biology, not in the computational treatment of the data."* (Unknown referee)
>
> *"In this age of specialization men who thoroughly know one field are often incompetent to discuss another."* (Richard Feynman)
>
> *"Science advances one funeral at a time."* (after Max Planck)

This brings us to the end of this textbook. I have tried to explain the details from mass spectrometry as cautiously as possible, and I hope that I have not put off too many readers from the computer science and bioinformatics side. I have also tried, but probably less successfully so, to explain the details from mathematics, combinatorics, and algorithmics as cautiously as possible; I hope that readers from the mass spectrometry side enjoyed reading the textbook. You have to excuse the tons of equations I unloaded on your head; but I am a trained mathematician.

I want to close this textbook with some personal thoughts on the field of computational mass spectrometry.

In 2005, I read an article by Sean R. Eddy [95] on "antedisciplinary science", which is "the science that precedes the organization of new disciplines, the Wild West frontier stage that comes before the law arrives." I do not agree with everything that Eddy writes in that article; but he has a point. Computer science is an example of a formerly antedisciplinary science, coming to life at the borders of electronics engineering and mathematics; bioinformatics still resides at the borders of biology and computer science; and as Eddy argues, the same is true for molecular biology.

Computational mass spectrometry is currently an interdisciplinary science and might, at some stage, turn into an antedisciplinary science: As Eddy points out, "new disciplines eventually self-organize around new problems and approaches, creating a new shared culture." With only three decades or so on its back, computational mass spectrometry still has some way to go: For proteomics, computational mass spectrometry is still in an early stage of becoming an antedisciplinary science; for metabolomics and glycomics, this transformation has hardly begun.

Negative attitudes of the type "it is different so it cannot be important", can be observed in any scientific field. But in general, bioinformatics is very open to novel topics, driven by new questions from biology as well as new data from biotechnology. Consequently, computational mass spectrometry is usually greeted rather positively by the bioinformatics community, potentially with a raised eyebrow saying, "do you really think that anyone is interested in that?" That is despite the fact that computational mass spectrometry is very different from the "beaten paths" in bioinformatics (sequence analysis, transcriptomics, cancer research etc.), in particular if we go beyond proteins and peptides: Metabolites and glycans are not even strings...

In my impression, this has been different in the MS community. For a long time — and potentially, even today — many people in mass spectrometry believed that the computational problems (which always boil down to "write a computer program", but who cares about the details) can be solved by vendors or postdocs.[1] Bioinformaticians, people who cannot even operate an MS instrument and often have a very

[1] I remember a particular occasion at a bioinformatics conference around 1998, where the invited speaker gave a talk on mass spectrometry, and the issues and questions for bioinformatics. He pretty much said that he wanted to pass the computational questions to some computer postdocs who worked in the basement of his lab (I am leaving out certain details as these might be regarded as "politically incorrect"); and that those questions will all be solved shortly. I am not sure if he talked about sequencing peptides or DNA; but in both cases, it appears that there were some major issues with his lab's basement.

basic understanding of biochemistry, were definitely not believed to contribute anything important to the field. Later, there was a strong belief that all the required software will come from the "major labs", as these know best where the true problems are, and they have "big teams" to solve them.

Many years ago, similar patterns were to be observed in the field of sequence analysis: Initially, biologists loathed the results of multiple sequence alignment programs; today, nobody would do an MSA completely by hand. Later, new companies promised to do the analysis; none of them exist anymore. Today, companies selling second-generation sequencing instruments, develop computational methods in close collaboration with the bioinformatics community. Over the last decades, queer and peculiar new approaches, data structures and algorithms were developed in the theoretical bioinformatics community, somewhat uncoupled from any biological application, a prominent example being compressed index structures. Today, compressed index structures are of utmost importance to handle the massive amount of data that new sequencing technologies are spitting out each day, and reside at the heart of many read mappers.

Even though I have spent more than twenty years in computational mass spectrometry, I do not consider myself an expert in mass spectrometry issues; in fact, I hardly ever sat in front of an MS instrument after my time in industry. So, whenever I approach a new problem in mass spectrometry, I do so in collaboration with an MS expert. If you happen to be an MS expert and you encounter a problem that has to do with the computational analysis of your data, you might want to do a similar thing: Collaborate with a computational mass spectrometry expert. If there is a fire, call the fire brigade. Not your postdoc.

For the "big team" issue, I fully agree with Eddy that computer science "rests more on the lasting visions of unique intellectual adventurers" than on the big teams it likes to mythologize: "Teams" in computer science are usually small — in 2010, about half of the papers listed in DBLP had one or two authors — and each computer scientist may play for many teams. A similar statement is true for most of bioinformatics: I just mention pairwise sequence alignments, Karlin-Altschul statistics and compressed index structures, which were surely not developed by large teams in large sequencing labs. I do not see any reason why for computational mass spectrometry, this should be different.

With that being said, I come to the end. The aim of this textbook was to explain some of the fundamental questions that happen to raise their heads over and over again in computational mass spectrometry. But clearly, this is extremely subjective. I hope that I have inspired the reader to deal with questions from this field which I, myself, have found both interesting and inspiring over the last twenty years, and which still keep me in their spell. Also, I hope that I have provided a small building block for what might once turn computational mass spectrometry into a truly antedisciplinary science. Maybe, this is not where the music plays; but then, let us bring in some instruments.

The full quote by Max Planck is, "A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it."

Jena, November 7, 2026

Bibliography

[1] F. Allen, R. Greiner and D. Wishart. Competitive fragmentation modeling of ESI-MS/MS spectra for putative metabolite identification. *Metabolomics*, 11(1):98–110, 2015.

[2] F. Allen, A. Pon, R. Greiner and D. Wishart. Computational prediction of electron ionization mass spectra to assist in GC/MS compound identification. *Anal Chem*, 88(15):7689–7697, 2016.

[3] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, 1997.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215:403–410, 1990.

[5] J. Anderson and V. Blair. Penalized maximum likelihood estimation in logistic regression and discrimination. *Biometrika*, 69(1):123–136, 1982.

[6] S. Andreotti, G. W. Klau and K. Reinert. Antilope – a Lagrangian relaxation approach to the *de novo* peptide sequencing problem. *IEEE/ACM Trans Comput Biology Bioinform*, 9(2):385–394, 2012.

[7] R. Apweiler, H. Hermjakob and N. Sharon. On the frequency of protein glycosylation, as deduced from analysis of the SWISS-PROT database. *Biochim Biophys Acta*, 1473(1):4–8, 1999.

[8] J.-M. Autebert, J. Berstel and L. Boasson. Context-free languages and pushdown automata. In: *Handbook of Formal Languages*. Ed. by G. Rozenberg and A. Salomaa. Vol. 1. Springer, Berlin, 1997, pages 111–174.

[9] A. Azulay and Y. Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *J Mach Learn Res*, 20(184):1–25, 2019.

[10] V. Bafna and N. Edwards. On de novo interpretation of tandem mass spectra for peptide identification. In: *Proc. of Research in Computational Molecular Biology (RECOMB 2003)*. 2003, pages 9–18.

[11] N. Bandeira, K. R. Clauser and P. A. Pevzner. Shotgun protein sequencing: assembly of peptide tandem mass spectra from mixtures of modified proteins. *Mol Cell Proteomics*, 6(7):1123–1134, 2007.

[12] C. Bartels. Fast algorithm for peptide sequencing by mass spectrometry. *Biomed Environ Mass Spectrom*, 19:363–368, 1990.

[13] J. M. S. Bartlett and D. Stirling. A short history of the polymerase chain reaction. *Methods Mol Biol*, 226:3–6, 2003.

[14] C. A. Bauer and S. Grimme. How to compute electron ionization mass spectra from first principles. *J Phys Chem A*, 120(21):3755–3766, 2016.

[15] M. Beck, I. M. Gessel and T. Komatsu. The polynomial part of a restricted partition function related to the Frobenius problem. *Electron J Comb*, 8(1):N7, 2001.

[16] D. E. Beihoffer, J. Hendry, A. Nijenhuis and S. Wagon. Faster algorithms for Frobenius numbers. *Electron J Comb*, 12:R27, 2005.

[17] C. Benecke, R. Grund, R. Hohberger, A. Kerber, R. Laue and T. Wieland. MOLGEN+, a generator of connectivity isomers and stereoisomers for molecular structure elucidation. *Anal Chim Acta*, 314:141–147, 1995.

[18] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J Roy Stat Soc B*, 57(1):289–300, 1995.

[19] G. Benson. Composition alignment. In: *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2003)*. Vol. 2812. Lect Notes Comput Sci. Springer, Berlin, 2003, pages 447–461.

[20] A. Bertsch, A. Leinenbach, A. Pervukhin, M. Lubeck, R. Hartmer, C. Baessmann, Y. A. Elnakady, R. Müller, S. Böcker, C. G. Huber, and O. Kohlbacher. De novo peptide sequencing by tandem MS using complementary CID and electron transfer dissociation. *Electrophoresis*, 30(21):3736–3747, 2009.

[21] K. Biemann, C. Cone and B. R. Webster. Computer-aided interpretation of high-resolution mass spectra. II. Amino acid sequence of peptides. *J Am Chem Soc*, 88(11):2597–2598, 1966.

[22] K. Biemann, C. Cone, B. R. Webster and G. P. Arsenault. Determination of the amino acid sequence in oligopeptides by computer interpretation of their high-resolution mass spectra. *J Am Chem Soc*, 88(23):5598–5606, 1966.

[23] W. Bittremieux, V. Ananth, W. E. Fondrie, C. Melendez, M. Pominova, J. Sanders, B. Wen, M. Yilmaz, and W. S. Noble. Deep learning methods for *De Novo* peptide sequencing. *Mass Spectrom Rev*, 45(3):507–526, 2026.

[24] A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto. Fourier meets Möbius: fast subset convolution. In: *Proc. of ACM Symposium on Theory of Computing (STOC 2007)*. ACM press, New York, 2007, pages 67–74.

[25] S. Böcker. Sequencing from compomers: using mass spectrometry for DNA de-novo sequencing of 200+ nt. *J Comput Biol*, 11(6):1110–1134, 2004.

[26] S. Böcker. Searching molecular structure databases using tandem MS data: are we there yet? *Curr Opin Chem Biol*, 36:1–6, 2017.

[27] S. Böcker, S. Briesemeister and G. W. Klau. Exact algorithms for cluster editing: evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

[28] S. Böcker and K. Dührkop. Fragmentation trees reloaded. *J Cheminformatics*, 8:5, 2016.

[29] S. Böcker, B. Kehr and F. Rasche. Determination of glycan structure from tandem mass spectra. *IEEE/ACM Trans Comput Biology Bioinform*, 8(4):976–986, 2011.

[30] S. Böcker, M. Letzel, Z. Lipták and A. Pervukhin. Decomposing metabolomic isotope patterns. In: *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2006)*. Vol. 4175. Lect Notes Comput Sci. Springer, Berlin, 2006, pages 12–23.

[31] S. Böcker, M. Letzel, Z. Lipták and A. Pervukhin. SIRIUS: decomposing isotope patterns for metabolite identification. *Bioinformatics*, 25(2):218–224, 2009.

[32] S. Böcker and Z. Lipták. Efficient mass decomposition. In: *Proc. of ACM Symposium on Applied Computing (SAC 2005)*. ACM press, New York, 2005, pages 151–157.

[33] S. Böcker and Z. Lipták. The Money Changing Problem revisited: computing the Frobenius number in time $O(k\, a_1)$. In: *Proc. of Computing and Combinatorics Conference (COCOON 2005)*. Vol. 3595. Lect Notes Comput Sci. Springer, Berlin, 2005, pages 965–974.

[34] S. Böcker and Z. Lipták. A fast and simple algorithm for the Money Changing Problem. *Algorithmica*, 48(4):413–432, 2007.

[35] S. Böcker and A. Pervukhin. Inferring peptide composition from molecular formulas. In: *Proc. of Computing and Combinatorics Conference (COCOON 2009)*. Vol. 5609. Lect Notes Comput Sci. Springer, Berlin, 2009, pages 277–286.

[36] S. Böcker and F. Rasche. Towards de novo identification of metabolites by analyzing tandem mass spectra. *Bioinformatics*, 24:I49–I55, 2008. Proc. of *European Conference on Computational Biology* (ECCB 2008).

[37] S. Böcker, F. Rasche and T. Steijger. Annotating fragmentation patterns. In: *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2009)*. Vol. 5724. Lect Notes Comput Sci. Springer, Berlin, 2009, pages 13–24.

[38] S. Böcker and S. Wagner. Counting glycans revisited. *J Math Biol*, 69(4):799–816, 2014.

[39] R. S. Bohacek, C. McMartin and W. C. Guida. The art and practice of structure-based drug design: a molecular modeling perspective. *Med Res Rev*, 16(1):3–50, 1996.

[40] A. Brauer and J. E. Shockley. On a problem of Frobenius. *J Reine Angew Math*, 211:215–220, 1962.

[41] R. Breitling, A. R. Pitt and M. P. Barrett. Precision mapping of the metabolome. *Trends Biotechnol*, 24(12):543–548, 2006.

[42] C. D. Broeckling, F. A. Afsar, S. Neumann, A. Ben-Hur and J. E. Prenni. RAMClust: a novel feature clustering method enables spectral-matching-based annotation for metabolomics data. *Anal Chem*, 86(14):6812–6817, 2014.

[43] C. Brouard, E. Bach, S. Böcker and J. Rousu. Magnitude-preserving ranking for structured outputs. In: *Proc. of Asian Conference on Machine Learning (ACML 2017)*. Vol. 77. Proceedings of Machine Learning Research. PMLR, 2017, pages 407–422.

[44] C. Brouard, H. Shen, K. Dührkop, F. d'Alché-Buc, S. Böcker and J. Rousu. Fast metabolite identification with input output kernel regression. *Bioinformatics*, 32(12):i28–i36, 2016. Proc. of *Intelligent Systems for Molecular Biology* (ISMB 2016).

[45] K. Q. Brown. Geometric transforms for fast geometric algorithms. Report CMUCS-80-101. Pittsburgh, USA: Dept. Comput. Sci., Carnegie-Mellon Univ., 1980.

[46] R. Bruderer, O. M. Bernhardt, T. Gandhi, S. M. Miladinović, L.-Y. Cheng, S. Messner, T. Ehrenberger, V. Zanotelli, Y. Butscheid, C. Escher, O. Vitek, O. Rinner, and L. Reiter. Extending the limits of quantitative proteome profiling with data-independent acquisition and application to acetaminophen-treated three-dimensional liver microtissues. *Mol Cell Proteomics*, 14(5):1400–1410, 2015.

[47] Y. Buehler, S. Javor and J.-L. Reymond. Sampling a trillion-sized GDB-20 database of drug-like molecules by generative artificial intelligence. *Digital Discovery*, 2026. In press.

[48] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals and M. Vingron. q-gram based database searching using a suffix array. In: *Proc. of Research in Computational Molecular Biology (RECOMB 1999)*. ACM press, New York, 1999, pages 77–83.

[49] S. Cappadona, P. Nanni, M. Benevento, F. Levander, P. Versura, A. Roda, S. Cerutti, and L. Pattini. Improved label-free LC-MS analysis by wavelet-based noise rejection. *J Biomed Biotechnol*, 2010:131505, 2010.

[50] J. Cautereels, M. Claeys, D. Geldof and F. Blockhuys. Quantum chemical mass spectrometry: ab initio prediction of electron ionization mass spectra and identification of new fragmentation pathways. *J Mass Spectrom*, 51(8):602–614, 2016.

[51] E. Cayley. Ueber die analytischen Figuren, welche in der Mathematik Bäume genannt werden und ihre Anwendung auf die Theorie chemischer Verbindungen. *Ber Dtsch Chem Ges*, 8(2):1056–1059, 1875.

[52] A. Ceroni, K. Maass, H. Geyer, R. Geyer, A. Dell and S. M. Haslam. GlycoWorkbench: a tool for the computer-assisted annotation of mass spectra of glycans. *J Proteome Res*, 7(4):1650–1659, 2008.

[53] T. Chari and L. Pachter. The specious art of single-cell genomics. *PLoS Comput Biol*, 19(8):e1011288, 2023.

[54] E. Check. Proteomics and cancer: running before we can walk? *Nature*, 429:496–497, 2004.

[55] T. Chen, M.-Y. Kao, M. Tepel, J. Rush and G. M. Church. A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. In: *Proc. of Symposium on Discrete Algorithms (SODA 2000)*. Society for Industrial and Applied Mathematics, 2000, pages 389–398.

[56] T. Chen, M.-Y. Kao, M. Tepel, J. Rush and G. M. Church. A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. *J Comput Biol*, 8(3):325–337, 2001.

[57] H. Choi, D. Ghosh and A. I. Nesvizhskii. Statistical validation of peptide identifications in large-scale proteomics using the target-decoy database search strategy and flexible mixture modeling. *J Proteome Res*, 7(1):286–292, 2008.

[58] H. Choi and A. I. Nesvizhskii. Semisupervised model-based validation of peptide identifications in mass spectrometry-based proteomics. *J Proteome Res*, 7(1):254–265, 2008.

[59] H. H. Chou, H. Takematsu, S. Diaz, J. Iber, E. Nickerson, K. L. Wright, E. A. Muchmore, D. L. Nelson, S. T. Warren, and A. Varki. A mutation in human CMP-sialic acid hydroxylase occurred after the Homo-Pan divergence. *Proc Natl Acad Sci U S A*, 95(20):11751–11756, 1998.

[60] S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. London: Springer, 2001.

[61] P. Compeau and P. Pevzner. *Bioinformatics algorithms: an active learning approach*. 3rd. Active Learning Publishers La Jolla, 2018.

[62] S. A. Cook. The complexity of theorem-proving procedures. In: *Proc. of ACM Symposium on Theory Of Computing (STOC 1971)*. ACM press, New York, 1971, pages 151–158.

[63] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math Comp*, 19(90):297–301, 1965.

[64] C. A. Cooper, E. Gasteiger and N. H. Packer. GlycoMod – a software tool for determining glycosylation compositions from mass spectrometric data. *Proteomics*, 1(2):340–349, 2001.

[65] T. H. Cormen, C. E. Leiserson and R. L. Rivest. *Introduction to Algorithms*. First. The MIT Press, 1990.

[66] R. Craig and R. C. Beavis. TANDEM: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, 2004.

[67] B. Curry and D. E. Rumelhart. MSnet: a neural network that classifies mass spectra. *Tetrahedron Com Methodol*, 3:213–237, 1990.

[68] R. R. da Silva, M. Wang, L.-F. Nothias, J. J. J. van der Hooft, A. M. Caraballo-Rodríguez, E. Fox, M. J. Balunas, J. L. Klassen, N. P. Lopes, and P. C. Dorrestein. Propagating annotations of molecular networks using in silico fragmentation. *PLoS Comput Biol*, 14(4):e1006089, 2018.

[69] V. Dančík, T. A. Addona, K. R. Clauser, J. E. Vath and P. A. Pevzner. De novo peptide sequencing via tandem mass spectrometry: a graph-theoretical approach. In: *Proc. of Research in Computational Molecular Biology (RECOMB 1999)*. 1999, pages 135–144.

[70] V. Dančík, T. A. Addona, K. R. Clauser, J. E. Vath and P. A. Pevzner. De novo peptide sequencing via tandem mass spectrometry: a graph-theoretical approach. *J Comput Biol*, 6(3/4):327–342, 1999.

[71] C. Dass. *Principles and practice of biological mass spectrometry*. John Wiley and Sons, 2001.

[72] A. C. Davison and D. V. Hinkley. *Bootstrap methods and their application*. Cambridge University Press, 1997.

[73] J. L. Davison. On the linear diophantine problem of Frobenius. *J Number Theory*, 48(3):353–363, 1994.

[74] M. O. Dayhoff, R. M. Schwartz and B. C. Orcutt. A model of evolutionary change in proteins. In: *Atlas of protein sequence and structure*. Vol. 5 suppl. 3. National Biomedical Research Foundation, 1978. Chap. 22, pages 345–351.

[75] H. E. Dayringer and F. W. McLafferty. Computer-aided interpretation of mass spectra (part xiv). STIRS prediction of rings-plus-double-bonds values. *Org Mass Spectrom*, 12(1):53–54, 1977.

[76] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. second. Springer, Berlin, 2000.

[77] E. de Hoffmann and V. Stroobant. *Mass Spectrometry: Principles and Applications*. third. Wiley-Interscience, 2007, page 502.

[78] J. R. de Laeter, J. K. Böhlke, P. D. Biévre, H. Hidaka, H. S. Peiser, K. J. R. Rosman and P. D. P. Taylor. Atomic weights of the elements. Review 2000 (IUPAC technical report). *Pure Appl Chem*, 75(6):683–800, 2003.

[79] S. Degroeve and L. Martens. MS2PIP: a tool for MS/MS peak intensity prediction. *Bioinformatics*, 29(24):3199–3203, 2013.

[80] V. Demichev, C. B. Messner, S. I. Vernardis, K. S. Lilley and M. Ralser. DIA-NN: neural networks and interference correction enable deep proteome coverage in high throughput. *Nat Methods*, 17(1):41–44, 2020.

[81] R. Diestel. *Graph Theory*. 4th. Vol. 173. Graduate Texts in Mathematics. Springer-Verlag, 2010.

[82] P. Dittwald, D. Valkenborg, J. Claesen, A. L. Rockwood and A. Gambin. On the fine isotopic distribution and limits to resolution in mass spectrometry. *J Am Soc Mass Spectrom*, 26(10):1732–1745, 2015.

[83] Y. Djoumbou Feunang. Cheminformatics tools for enabling metabolomics. PhD thesis. Department of Biological Sciences, University of Alberta, 2017.

[84] B. Domon and C. E. Costello. A systematic nomenclature for carbohydrate fragmentations in FAB-MS/MS spectra of glycoconjugates. *Glycoconjugate J*, 5:397–409, 1988.

[85] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, Berlin, 1999.

[86] H. L. Dreyfus and S. E. Dreyfus. What artificial experts can and cannot do. *AI & society*, 6(1):18–26, 1992.

[87] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1972.

[88] P. Du, W. A. Kibbe and S. Lin. Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, 22(17):2059–2065, 2006.

[89] K. Dührkop. Computational methods for small molecule identification. PhD thesis. Jena, Germany: Friedrich-Schiller-Universität Jena, 2018.

[90] K. Dührkop, M. Fleischauer, M. Ludwig, A. A. Aksenov, A. V. Melnik, M. Meusel, P. C. Dorrestein, J. Rousu, and S. Böcker. SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information. *Nat Methods*, 16(4):299–302, 2019.

[91] K. Dührkop, M. A. Lataretu, W. T. J. White and S. Böcker. Heuristic algorithms for the maximum colorful subtree problem. In: *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2018)*. Vol. 113. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 23:1–23:14.

[92] K. Dührkop, M. Ludwig, M. Meusel and S. Böcker. Faster mass decomposition. In: *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2013)*. Vol. 8126. Lect Notes Comput Sci. Springer, Berlin, 2013, pages 45–58.

[93] K. Dührkop, H. Shen, M. Meusel, J. Rousu and S. Böcker. Searching molecular structure databases with tandem mass spectra using CSI:FingerID. *Proc Natl Acad Sci U S A*, 112(41):12580–12585, 2015.

[94] M. Dyer. Approximate counting by dynamic programming. In: *Proc. of Symposium on Theory of Computing (STOC 2003)*. 2003, pages 693–699.

[95] S. R. Eddy. "antedisciplinary" science. *PLoS Comput Biol*, 1(1):e6, 2005.

[96] P. Edman. Method for determination of the amino acid sequence in peptides. *Acta Chem Scand*, 4:283–293, 1950.

[97] B. Efron and R. Tibshirani. Empirical bayes methods and false discovery rates for microarrays. *Genet Epidemiol*, 23(1):70–86, 2002.

[98] B. Efron, R. Tibshirani, J. D. Storey and V. Tusher. Empirical Bayes analysis of a microarray experiment. *J Am Stat Assoc*, 96(456):1151–1160, 2001.

[99] M. Ehrich, S. Böcker and D. van den Boom. Multiplexed discovery of sequence polymorphisms using base-specific cleavage and MALDI-TOF MS. *Nucleic Acids Res*, 33(4):e38, 2005.

[100] D. Einstein, D. Lichtblau, A. Strzebonski and S. Wagon. Frobenius numbers by lattice point enumeration. *INTEGERS*, 7(1):#A15, 2007.

[101] J. E. Elias and S. P. Gygi. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nat Methods*, 4(3):207–214, 2007.

[102] J. E. Elias, F. D. Gibbons, O. D. King, F. P. Roth and S. P. Gygi. Intensity-based protein identification by machine learning from a library of tandem mass spectra. *Nat Biotechnol*, 22(2):214–219, 2004.

[103] J. K. Eng, A. L. McCormack and J. R. Yates III. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *J Am Soc Mass Spectrom*, 5(11):976–989, 1994.

[104] D. Eppstein. Finding the k shortest paths. In: *Proc. of Foundations of Computer Science (FOCS 1994)*. 1994, pages 154–165.

[105] D. Eppstein, Z. Galil, R. Giancarlo and G. F. Italiano. Sparse dynamic programming I: linear cost functions. *J ACM*, 39(3):519–545, 1992.

[106] M. Ethier, J. A. Saba, M. Spearman, O. Krokhin, M. Butler, W. Ens, K. G. Standing, and H. Perreault. Application of the StrOligo algorithm for the automated structure assignment of complex N-linked glycans from glycoproteins using tandem mass spectrometry. *Rapid Commun Mass Spectrom*, 17(24):2713–2720, 2003.

[107] J.-L. Faulon, D. P. Visco Jr and D. Roe. Enumerating molecules. *Reviews in Computational Chemistry*, 21:209–286, 2005.

[108] M. Fellows, G. Fertin, D. Hermelin and S. Vialette. Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. In: *Proc. of International Colloquium on Automata, Languages and Programming (ICALP 2007)*. Vol. 4596. Lect Notes Comput Sci. Springer, Berlin, 2007, pages 340–351.

[109] J. Fenn, M. Mann, C. Meng, S. Wong and C. Whitehouse. Electrospray ionisation for mass spectrometry of large biomolecules. *Science*, 246(4926):64–71, 1989.

[110] D. Fenyö and R. C. Beavis. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. *Anal Chem*, 75(4):768–774, 2003.

[111] J. Fernandez-de-Cossio Diaz and J. Fernandez-de-Cossio. Computation of isotopic peak center-mass distribution by Fourier transform. *Anal Chem*, 84(16):7052–7056, 2012.

[112] A. R. Fernie, R. N. Trethewey, A. J. Krotzky and L. Willmitzer. Metabolite profiling: from diagnostics to systems biology. *Nat Rev Mol Cell Biol*, 5(9):763–769, 2004.

[113] I. Ferrer and E. M. Thurman. Importance of the electron mass in the calculations of exact mass by time-of-flight mass spectrometry. *Rapid Commun Mass Spectrom*, 21(15):2538–2539, 2007.

[114] G. Fertin, J. Fradin and G. Jean. Algorithmic aspects of the maximum colorful arborescence problem. In: *Proc. of Theory and Applications of Models of Computation (TAMC 2017)*. Vol. 10185. Lect Notes Comput Sci. 2017, pages 216–230.

[115] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Freely available from http://algo.inria.fr/flajolet/Publications/book.pdf. Cambridge University Press, 2009, page 824.

[116] A. Frank and P. Pevzner. PepNovo: de novo peptide sequencing via probabilistic network modeling. *Anal Chem*, 77(4):964–973, 2005.

[117] V. A. Fusaro, D. R. Mani, J. P. Mesirov and S. A. Carr. Prediction of high-responding peptides for targeted protein assays by mass spectrometry. *Nat Biotechnol*, 27(2):190–198, 2009.

[118] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979, page 340.

[119] J. Gasteiger, W. Hanebeck and K.-P. Schulz. Prediction of mass spectra from structural information. *J Chem Inf Comput Sci*, 32(4):264–271, 1992.

[120] L. Gatto, K. D. Hansen, M. R. Hoopmann, H. Hermjakob, O. Kohlbacher and A. Beyer. Testing and validation of computational methods for mass spectrometry. *J Proteome Res*, 15(3):809–814, 2016.

[121] S. P. Gaucher, J. Morrow and J. A. Leary. STAT: a saccharide topology analysis tool used in combination with tandem mass spectrometry. *Anal Chem*, 72(11):2331–2336, 2000.

[122] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. Xu, D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant. Open mass spectrometry search algorithm. *J Proteome Res*, 3(5):958–964, 2004.

[123] M. Gerlich and S. Neumann. MetFusion: integration of compound identification strategies. *J Mass Spectrom*, 48(3):291–298, 2013.

[124] S. Gessulat, T. Schmidt, D. P. Zolg, P. Samaras, K. Schnatbaum, J. Zerweck, T. Knaute, J. Rechenberger, B. Delanghe, A. Huhmer, U. Reimer, H.-C. Ehrlich, S. Aiche, B. Kuster, and M. Wilhelm. Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nat Methods*, 16(6):509–518, 2019.

[125] L. C. Gillet, P. Navarro, S. Tate, H. Röst, N. Selevsek, L. Reiter, R. Bonner, and R. Aebersold. Targeted data extraction of the MS/MS spectra generated by data-independent acquisition: a new concept for consistent and accurate proteome analysis. *Mol Cell Proteomics*, 11(6):O111.016717, 2012.

[126] P. Gilmore and R. Gomory. Multi-stage cutting stock problems of two and more dimensions. *Oper Res*, 13(1):94–120, 1965.

[127] D. Goldberg, M. W. Bern, B. Li and C. B. Lebrilla. Automatic determination of O-glycan structure from fragmentation spectra. *J Proteome Res*, 5(6):1429–1434, 2006.

[128] D. Goldberg, M. W. Bern, S. J. North, S. M. Haslam and A. Dell. Glycan family analysis for deducing N-glycan topology from single MS. *Bioinformatics*, 25(3):365–371, 2009.

[129] D. Goldberg, M. W. Bern, S. Parry, M. Sutton-Smith, M. Panico, H. R. Morris and A. Dell. Automated N-glycopeptide identification using a combination of single- and tandem-MS. *J Proteome Res*, 6(10):3995–4005, 2007.

[130] D. Goldberg, M. Sutton-Smith, J. Paulson and A. Dell. Automatic annotation of matrix-assisted laser desorption/ionization N-glycan spectra. *Proteomics*, 5(4):865–875, 2005.

[131] S. Goldman, J. Li and C. W. Coley. Generating molecular fragmentation graphs with autoregressive neural networks. *Anal Chem*, 96(8):3419–3428, 2024.

[132] N. A. B. Gray, A. Buchs, D. H. Smith and C. Djerassi. Computer assisted structural interpretation of mass spectral data. *Helv Chim Acta*, 64(2):458–470, 1981.

[133] N. A. B. Gray, R. E. Carhart, A. Lavanchy, D. H. Smith, T. Varkony, B. G. Buchanan, W. C. White, and L. Creary. Computerized mass spectrum prediction and ranking. *Anal Chem*, 52(7):1095–1102, 1980.

[134] N. A. Gray. Applications of artificial intelligence for organic chemistry: analysis of C-13 spectra. *Artif Intell*, 22(1):1–21, 1984.

[135] H. Greenberg. Solution to a linear diophantine equation for nonnegative integers. *J Algorithms*, 9(3):343–353, 1988.

[136] D. H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms*. Vol. 1. Progress in Computer Science and Applied Logic (PCS). Birkhäuser Boston, 1990, pages 1–10.

[137] S. Grimme. Towards first principles calculation of electron impact mass spectra of molecules. *Angew Chem Int Ed Engl*, 52(24):6306–6312, 2013.

[138] J. H. Gross. *Mass Spectrometry: A textbook*. 3rd. Springer, Berlin, 2017.

[139] T. Grüner. Strategien zur konstruktion diskreter strukturen und ihre anwendung auf molekulare graphen. *MATCH Commun Math Comput Chem*, 39:39–126, 1999.

[140] M. Guilhaus. Principles and instrumentation in time-of-flight mass spectrometry. *J Mass Spectrom*, 30(11):1519–1532, 1995.

[141] S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. In: *Proc. of Mathematical Foundations of Computer Science (MFCS 2010)*. Vol. 6281. Lect Notes Comput Sci. Springer, Berlin, 2010, pages 405–416.

[142] V. D. Hähnke, S. Kim and E. E. Bolton. PubChem chemical structure standardization. *J Cheminformatics*, 10(1):36, 2018.

[143] C. Hamm, W. Wilson and D. Harvan. Peptide sequencing program. *Comput Appl Biosci*, 2:115–118, 1986.

[144] K. S. Haraki, R. Venkataraghavan and F. W. McLafferty. Prediction of substructures from unknown mass spectra by the self-training interpretive and retrieval system. *Anal Chem*, 53(3):386–392, 1981.

[145] F. Harary, R. W. Robinson and A. J. Schwenk. Twenty-step algorithm for determining the asymptotic number of trees of various species. *J Austral Math Soc*, 20(Series A):483–503, 1975.

[146] J. Hartler, A. Triebl, A. Ziegl, M. Trötzmüller, G. N. Rechberger, O. A. Zeleznik, K. A. Zierler, F. Torta, A. Cazenave-Gassiot, M. R. Wenk, A. F. C. E. Wheelock, A. M. Armando, O. Quehenberger, Q. Zhang, M. J. O. Wakelam, G. Haemmerle, F. Spener, H. C. Köfeler, and G. G. Thallinger. Deciphering lipid structures based on platform-independent decision rules. *Nat Methods*, 14(12):1171–1174, 2017.

[147] M. Heinonen, A. Rantanen, T. Mielikäinen, J. Kokkonen, J. Kiuru, R. A. Ketola and J. Rousu. FiD: a software for ab initio structural identification of product ions from tandem mass spectrometric data. *Rapid Commun Mass Spectrom*, 22(19):3043–3052, 2008.

[148] M. Heinonen, A. Rantanen, T. Mielikäinen, E. Pitkänen, J. Kokkonen and J. Rousu. Ab initio prediction of molecular fragments from tandem mass spectrometry data. In: *Proc. of German Conference on Bioinformatics (GCB 2006)*. Vol. P-83. Lecture Notes in Informatics. 2006, pages 40–53.

[149] M. Heinonen, H. Shen, N. Zamboni and J. Rousu. Metabolite identification and molecular fingerprint prediction via machine learning. *Bioinformatics*, 28(18):2333–2341, 2012.

[150] A. W. Hill and R. J. Mortishire-Smith. Automated assignment of high-resolution collisionally activated dissociation mass spectra using a systematic bond disconnection approach. *Rapid Commun Mass Spectrom*, 19(21):3111–3118, 2005.

[151] D. W. Hill, T. M. Kertesz, D. Fontaine, R. Friedman and D. F. Grant. Mass spectral metabonomics beyond elemental formula: chemical database querying by matching experimental with computational fragmentation spectra. *Anal Chem*, 80(14):5574–5582, 2008.

[152] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput*, 9(8):1735–1780, 1997.

[153] M. A. Hoffmann, F. Kretschmer, M. Ludwig and S. Böcker. Mad Hatter correctly annotates 98% of small molecule tandem mass spectra searching in PubChem. *Metabolites*, 13(3):314, 2023.

[154] P. Hong, H. Sun, L. Sha, Y. Pu, K. Khatri, X. Yu, Y. Tang, and C. Lin. GlycoDeNovo – an efficient algorithm for accurate de novo glycan topology reconstruction from tandem mass spectra. *J Am Soc Mass Spectrom*, 28(11):2288–2301, 2017.

[155] C. S. Hsu. Diophantine approach to isotopic abundance calculations. *Anal Chem*, 56(8):1356–1361, 1984.

[156] Q. Hu, R. J. Noll, H. Li, A. Makarov, M. Hardman and R. G. Cooks. The Orbitrap: a new mass spectrometer. *J Mass Spectrom*, 40(4):430–443, 2005.

[157] F. Hufsky and S. Böcker. Mining molecular structure databases: identification of small molecules based on fragmentation mass spectrometry data. *Mass Spectrom Rev*, 36(5):624–633, 2017.

[158] F. Hufsky, K. Scheubert and S. Böcker. Computational mass spectrometry for small molecule fragmentation. *Trends Anal Chem*, 53:41–48, 2014.

[159] F. Hufsky, K. Scheubert and S. Böcker. New kids on the block: Novel informatics methods for natural product discovery. *Nat Prod Rep*, 31(6):807–817, 2014.

[160] J. P. A. Ioannidis. Why most published research findings are false. *PLoS Med*, 2(8):e124, 2005.

[161] A. Ipsen. Efficient calculation of exact fine structure isotope patterns via the multidimensional Fourier transform. *Anal Chem*, 86(11):5316–5322, 2014.

[162] N. Jaitly, M. E. Monroe, V. A. Petyuk, T. R. W. Clauss, J. N. Adkins and R. D. Smith. Robust algorithm for alignment of liquid chromatography-mass spectrometry analyses in an accurate mass and time tag data analysis pipeline. *Anal Chem*, 78(21):7397–7409, 2006.

[163] T. Jebara, R. Kondor and A. Howard. Probability product kernels. *J Mach Learn Res*, 5:819–844, 2004.

[164] K. Jeong, S. Kim and N. Bandeira. False discovery rates in spectral identification. *BMC Bioinformatics*, 13 Suppl 16:S2, 2012.

[165] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In: *Proc. of Mathematical Foundations of Computer Science (MFCS 1991)*. Vol. 520. Lect Notes Comput Sci. Springer, Berlin, 1991, pages 240–248.

[166] H. J. Joshi, M. J. Harrison, B. L. Schulz, C. A. Cooper, N. H. Packer and N. G. Karlsson. Development of a mass fingerprinting tool for automated interpretation of oligosaccharide fragmentation data. *Proteomics*, 4(6):1650–1664, 2004.

[167] L. Käll, J. D. Canterbury, J. Weston, W. S. Noble and M. J. MacCoss. Semi-supervised learning for peptide identification from shotgun proteomics datasets. *Nat Methods*, 4(11):923–925, 2007.

[168] L. Käll, J. D. Storey, M. J. MacCoss and W. S. Noble. Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. *J Proteome Res*, 7(1):29–34, 2008.

[169] L. Käll, J. D. Storey, M. J. MacCoss and W. S. Noble. Posterior error probabilities and false discovery rates: two sides of the same coin. *J Proteome Res*, 7(1):40–44, 2008.

[170] L. Käll, J. D. Storey and W. S. Noble. Non-parametric estimation of posterior error probabilities associated with peptides identified by tandem mass spectrometry. *Bioinformatics*, 24(16):i42–i48, 2008.

[171] M. Kanehisa, S. Goto, M. Hattori, K. F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. Araki, and M. Hirakawa. From genomics to chemical genomics: new developments in KEGG. *Nucleic Acids Res*, 34:D354–D357, 2006.

[172] R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12(2):161–177, 1992.

[173] M. Karas and F. Hillenkamp. Laser desorption ionization of proteins with molecular masses exceeding 10,000 Daltons. *Anal Chem*, 60(20):2299–2301, 1988.

[174] S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci U S A*, 87(6):2264–2268, 1990.

[175] R. M. Karp. Reducibility among combinatorial problems. In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. New York: Plenum Press, 1972, pages 85–103.

[176] U. Keich, A. Kertesz-Farkas and W. S. Noble. Improved false discovery rate estimation procedure for shotgun proteomics. *J Proteome Res*, 14(8):3148–3161, 2015.

[177] U. Keich, M. Li, B. Ma and J. Tromp. On spaced seeds for similarity search. *Discrete Appl Math*, 138(3):253–263, 2004.

[178] U. Keich and W. S. Noble. On the importance of well-calibrated scores for identifying shotgun proteomics spectra. *J Proteome Res*, 14(2):1147–1160, 2015.

[179] U. Keich and W. S. Noble. Controlling the FDR in imperfect matches to an incomplete database. *J Am Stat Assoc*, 113(523):973–982, 2018.

[180] U. Keich, K. Tamura and W. S. Noble. Averaging strategy to reduce variability in target-decoy estimates of false discovery rate. *J Proteome Res*, 18(2):585–593, 2019.

[181] A. Keller, A. I. Nesvizhskii, E. Kolker and R. Aebersold. Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search. *Anal Chem*, 74(20):5383–5392, 2002.

[182] E. Kendrick. A mass scale based on CH2 = 14.0000 for high resolution mass spectrometry of organic compounds. *Anal Chem*, 35(13):2146–2154, 1963.

[183] A. Kerber, R. Laue, M. Meringer and C. Rücker. Molecules in silico: the generation of structural formulae and its applications. *J Comput Chem Japan*, 3(3):85–96, 2004.

[184] A. Kerber, R. Laue and D. Moser. Ein Strukturgenerator für molekulare Graphen. *Anal Chim Acta*, 235:221–228, 1990.

[185] A. Kerber, R. Laue, M. Meringer and C. Rücker. Molecules in silico: potential versus known organic compounds. *MATCH Commun Math Comput Chem*, 54(2):301–312, 2005.

[186] A. Kerber, R. Laue, M. Meringer, C. Rücker and E. Schymanski. *Mathematical Chemistry and Chemoinformatics: Structure Generation, Elucidation and Quantitative Structure-Property Relationships*. Berlin, Boston: De Gruyter, 2013.

[187] S. Kim, N. Gupta and P. A. Pevzner. Spectral probabilities and generating functions of tandem mass spectra: a strike against decoy databases. *J Proteome Res*, 7(8):3354–3363, 2008.

[188] S. Kim, I. Koo, X. Wei and X. Zhang. A method of finding optimal weight factors for compound identification in gas chromatography-mass spectrometry. *Bioinformatics*, 28(8):1158–1163, 2012.

[189] T. Kind and O. Fiehn. Metabolomic database annotations via query of elemental compositions: mass accuracy is insufficient even at less than 1 ppm. *BMC Bioinformatics*, 7(1):234, 2006.

[190] T. Kind and O. Fiehn. Seven golden rules for heuristic filtering of molecular formulas obtained by accurate mass spectrometry. *BMC Bioinformatics*, 8:105, 2007.

[191] T. Kind, K.-H. Liu, D. Y. Lee, B. DeFelice, J. K. Meissen and O. Fiehn. LipidBlast in silico tandem mass spectrometry database for lipid identification. *Nat Methods*, 10(8):755–758, 2013.

[192] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2005.

[193] P. Kreitzberg, J. Pennington, K. Lucke and O. Serang. Fast exact computation of the *k* most abundant isotope peaks with layer-ordered heaps. *Anal Chem*, 92(15):10613–10619, 2020.

[194] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc Amer Math Soc*, 7(1):48–50, 1956.

[195] H. Kubinyi. Calculation of isotope distributions in mass spectrometry: a trivial solution for a non-trivial problem. *Anal Chim Acta*, 247:107–119, 1991.

[196] K.-S. Kwok, R. Venkataraghavan and F. W. McLafferty. Computer-aided interpretation of mass spectra. III. Self-training interpretive and retrieval system. *J Am Chem Soc*, 95(13):4185–4194, 1973.

[197] M. K. Łącki, M. Startek, D. Valkenborg and A. Gambin. IsoSpec: hyperfast fine structure calculator. *Anal Chem*, 89(6):3272–3277, 2017.

[198] M. K. Łącki, D. Valkenborg and M. P. Startek. IsoSpec2: ultrafast fine structure calculator. *Anal Chem*, 92(14):9472–9475, 2020.

[199] V. Lacroix, C. G. Fernandes and M.-F. Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM Trans Comput Biology Bioinform*, 3(4):360–368, 2006.

[200] A. J. Lapadula, P. J. Hatcher, A. J. Hanneman, D. J. Ashline, H. Zhang and V. N. Reinhold. Congruent strategies for carbohydrate sequencing. 3. OSCAR: an algorithm for assigning oligosaccharide topology from MS^n data. *Anal Chem*, 77(19):6271–6279, 2005.

[201] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek and K.-R. Müller. Unmasking clever Hans predictors and assessing what machines really learn. *Nat Commun*, 10:1096, 2019.

[202] R. L. Last, A. D. Jones and Y. Shachar-Hill. Towards the plant metabolome and beyond. *Nat Rev Mol Cell Biol*, 8:167–174, 2007.

[203] A. Lavanchy, T. Varkony, D. H. Smith, N. A. B. Gray, W. C. White, R. E. Carhart, B. G. Buchanan, and C. Djerassi. Rule-based mass spectrum prediction and ranking: applications to structure elucidation of novel marine sterols. *Org Mass Spectrom*, 15(7):355–366, 1980.

[204] Y. LeCun, Y. Bengio and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[205] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput*, 1(4):541–551, 1989.

[206] J. Lederberg. Topological mapping of organic molecules. *Proc Natl Acad Sci U S A*, 53(1):134–139, 1965.

[207] J. Lederberg. How DENDRAL was conceived and born. In: *Proc. of ACM Conference on the History of Medical Informatics (HMI 1987)*. 1987, pages 5–19.

[208] R. C. Lee, R. L. Feinbaum and V. Ambros. The C. elegans heterochronic gene lin-4 encodes small RNAs with antisense complementarity to lin-14. *Cell*, 75(5):843–854, 1993.

[209] T. A. Lee. *A Beginner's Guide to Mass Spectral Interpretation*. Wiley, 1998, page 186.

[210] M. Lefmann, C. Honisch, S. Böcker, N. Storm, F. von Wintzingerode, C. Schlötelburg, A. Moter, D. van den Boom, and U. B. Göbel. A novel mass spectrometry based tool for genotypic identification of mycobacteria. *J Clin Microbiol*, 42(1):339–346, 2004.

[211] L. I. Levitsky, M. V. Ivanov, A. A. Lobas and M. V. Gorshkov. Unbiased false discovery rate estimation for shotgun proteomics based on the target-decoy approach. *J Proteome Res*, 16(2):393–397, 2017.

[212] G. Li and F. Ruskey. The advantages of forward thinking in generating rooted and free trees. In: *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*. Baltimore, Maryland, United States: Society for Industrial and Applied Mathematics, 1999, pages 939–940.

[213] L. Li, J. A. Kresh, N. M. Karabacak, J. S. Cobb, J. N. Agar and P. Hong. A hierarchical algorithm for calculating the isotopic fine structures of molecules. *J Am Soc Mass Spectrom*, 19(12):1867–1874, 2008.

[214] D. V. Lindley. *Making Decisions*. 2nd. London: Wiley, 1985.

[215] K. K. Lohmann and C.-W. von der Lieth. GlycoFragment and GlycoSearchMS: web tools to support the interpretation of mass spectra of complex carbohydrates. *Nucleic Acids Res*, 32(Web Server issue):W261–W266, 2004.

[216] M. Loos, C. Gerber, F. Corona, J. Hollender and H. Singer. Accelerated isotope fine structure calculation using pruned transition trees. *Anal Chem*, 87(11):5738–5744, 2015.

[217] R. Lou, Y. Cao, S. Li, X. Lang, Y. Li, Y. Zhang and W. Shui. Benchmarking commonly used software suites and analysis workflows for DIA proteomics and phosphoproteomics. *Nat Commun*, 14:94, 2023.

[218] R. Lou and W. Shui. Acquisition and analysis of DIA-based proteomic data: a comprehensive survey in 2023. *Mol Cell Proteomics*, 23(2):100712, 2024.

[219] B. Lu and T. Chen. A suffix tree approach to the interpretation of tandem mass spectra: applications to peptides of non-specific digestion and post-translational modifications. *Bioinformatics*, 19(Suppl 2):ii113–ii121, 2003. Proc. of *European Conference on Computational Biology* (ECCB 2003).

[220] M. Ludwig, L.-F. Nothias, K. Dührkop, I. Koester, M. Fleischauer, M. A. Hoffmann, D. Petras, F. Vargas, M. Morsy, L. Aluwihare, P. C. Dorrestein, and S. Böcker. Database-independent molecular formula annotation using Gibbs sampling through ZODIAC. *Nat Mach Intell*, 2(10):629–641, 2020.

[221] G. S. Lueker. Two NP-complete problems in nonnegative integer programming. Tech. rep. TR-178. 1975.

[222] Y.-R. Luo. *Handbook of Bond Dissociation Energies in Organic Compounds*. Boca Raton: CRC Press, 2003, page 392.

[223] B. Ma. Novor: real-time peptide de novo sequencing software. *J Am Soc Mass Spectrom*, 26(11):1885–1894, 2015.

[224] B. Ma and R. Johnson. De novo sequencing and homology searching. *Mol Cell Proteomics*, 11(2):O111.014902, 2012.

[225] B. Ma, J. Tromp and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[226] B. Ma, K. Zhang, C. Hendrie, C. Liang, M. Li, A. Doherty-Kirby and G. Lajoie. PEAKS: powerful software for peptide de novo sequencing by tandem mass spectrometry. *Rapid Commun Mass Spectrom*, 17(20):2337–2342, 2003.

[227] B. Ma, K. Zhang and C. Liang. An effective algorithm for peptide de novo sequencing from MS/MS spectra. *J Comput System Sci*, 70:418–430, 2005.

[228] K. Maass, R. Ranzinger, H. Geyer, C.-W. von der Lieth and R. Geyer. "Glyco-peakfinder" — *de novo* composition analysis of glycoconjugates. *Proteomics*, 7(24):4435–4444, 2007.

[229] P. Mallick, M. Schirle, S. S. Chen, M. R. Flory, H. Lee, D. Martin, J. Ranish, B. Raught, R. Schmitt, T. Werner, B. Kuster, and R. Aebersold. Computational prediction of proteotypic peptides for quantitative proteomics. *Nat Biotechnol*, 25(1):125–131, 2007.

[230] S. Martello and P. Toth. An exact algorithm for large unbounded knapsack problems. *Oper Res Lett*, 9(1):15–20, 1990.

[231] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Chichester: John Wiley & Sons, 1990.

[232] V. Marx. Machine learning, practically speaking. *Nat Methods*, 16(6):463–467, 2019.

[233] J. W. May and C. Steinbeck. Efficient ring perception for the Chemistry Development Kit. *J Cheminformatics*, 6(1):3, 2014.

[234] J. Mayer, K. Khairy and J. Howard. Drawing an elephant with four complex parameters. *Am J Phys*, 78(6):648–649, 2010.

[235] R. T. McCoy, E. Pavlick and T. Linzen. Right for the wrong reasons: diagnosing syntactic heuristics in natural language inference. 2019.

[236] A. D. McEachran, K. Mansouri, C. Grulke, E. L. Schymanski, C. Ruttkies and A. J. Williams. "MS-Ready" structures for non-targeted high-resolution mass spectrometry screening studies. *J Cheminformatics*, 10(1):45, 2018.

[237] B. D. McKay, M. A. Yirik and C. Steinbeck. Surge: a fast open-source chemical graph generator. *J Cheminf*, 14(1):24, 2022.

[238] J. Meija, T. B. Coplen, M. Berglund, W. A. Brand, P. De Bièvre, M. Gröning, N. E. Holden, J. Irrgeher, R. D. Loss, T. Walczyk, and T. Prohaska. Atomic weights of the elements 2013 (IUPAC technical report). *Pure Appl Chem*, 88(3):265–291, 2016.

[239] J. Meija, T. B. Coplen, M. Berglund, W. A. Brand, P. De Bièvre, M. Gröning, N. E. Holden, J. Irrgeher, R. D. Loss, T. Walczyk, and T. Prohaska. Isotopic compositions of the elements 2013 (IUPAC technical report). *Pure Appl Chem*, 88(3):293–306, 2016.

[240] P. E. Miller and M. B. Denton. The quadrupole mass filter: basic operating concepts. *J Chem Educ*, 63:617–622, 1986.

[241] J. Milton. *Tramp: The Life of Charlie Chaplin*. Da Capo Press, 1998.

[242] L. Mo, D. Dutta, Y. Wan and T. Chen. MSNovo: a dynamic programming algorithm for de novo peptide sequencing via tandem mass spectrometry. *Anal Chem*, 79(13):4870–4878, 2007.

[243] R. E. Moore, M. K. Young and T. D. Lee. Qscore: an algorithm for evaluating SEQUEST database search results. *J Am Soc Mass Spectrom*, 13(4):378–386, 2002.

[244] E. Mostacci, C. Truntzer, H. Cardot and P. Ducoroy. Multivariate denoising methods combining wavelets and principal component analysis for mass spectrometry data. *Proteomics*, 10(14):2564–2572, 2010.

[245] Z. Mucsi, B. Viskolcz and I. G. Csizmadia. A quantitative scale for the degree of aromaticity and antiaromaticity: a comparison of theoretical and experimental enthalpies of hydrogenation. *J Phys Chem A*, 111(6):1123–1132, 2007.

[246] I. K. Mun and F. W. Mclafferty. Computer methods of molecular structure elucidation from unknown mass spectra. In: *Supercomputers in Chemistry*. ACS Symposium Series. American Chemical Society, 1981. Chap. 9, pages 117–124.

[247] T. Muth, F. Hartkopf, M. Vaudel and B. Y. Renard. A potential golden age to come-current tools, recent use cases, and future avenues for de novo sequencing in proteomics. *Proteomics*, 18(18):e1700150, 2018.

[248] T. Muth and B. Y. Renard. Evaluating de novo sequencing in proteomics: already an accurate alternative to database-driven peptide identification? *Briefings Bioinf*, 19(5):954–970, 2018.

[249] A. I. Nesvizhskii, A. Keller, E. Kolker and R. Aebersold. A statistical model for identifying proteins by tandem mass spectrometry. *Anal Chem*, 75(17):4646–4658, 2003.

[250] A. I. Nesvizhskii. A survey of computational methods and error rate estimation procedures for peptide and protein identification in shotgun proteomics. *J Proteomics*, 73(11):2092–2123, 2010.

[251] D. H. Nguyen, C. H. Nguyen and H. Mamitsuka. Recent advances and prospects of computational methods for metabolite identification: a review with emphasis on machine learning approaches. *Briefings Bioinf*, 2018. Advance Access Publication, doi 10.1093/bib/bby066.

[252] N. Nguyen, H. Huang, S. Oraintara and A. Vo. Mass spectrometry data processing using zero-crossing lines in multi-scale of Gaussian derivative wavelet. *Bioinformatics*, 26(18):i659–i665, 2010.

[253] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[254] T. Nishioka, T. Kasama, T. Kinumi, H. Makabe, F. Matsuda, D. Miura, M. Miyashita, T. Nakamura, K. Tanaka, and A. Yamamoto. Winners of CASMI2013: automated tools and challenge data. *Mass Spectrom*, 3(special issue 2):S0039, 2014.

[255] T. Niven and H.-Y. Kao. Probing neural network comprehension of natural language arguments. 2019.

[256] W. S. Noble. Mass spectrometrists should search only for peptides they care about. *Nat Methods*, 12(7):605–608, 2015.

[257] L.-F. Nothias, D. Petras, R. Schmid, K. Dührkop, J. Rainer, A. Sarvepalli, I. Protsyuk, M. Ernst, H. Tsugawa, M. Fleischauer, F. Aicheler, A. A. Aksenov, O. Alka, P.-M. Allard, A. Barsch, X. Cachet, A. M. Caraballo-Rodríguez, R. R. Da Silva, T. Dang, N. Garg, J. M. Gauglitz, A. Gurevich, G. Isaac, A. K. Jarmusch, Z. Kameník, K. B. Kang, N. Kessler, I. Koester, A. Korf, A. Le Gouellec, M. Ludwig, C. Martin H, L.-I. McCall, J. McSayles, S. W. Meyer, H. Mohimani, M. Morsy, O. Moyne, S. Neumann, H. Neuweger, N. H. Nguyen, M. Nothias-Esposito, J. Paolini, V. V. Phelan, T. Pluskal, R. A. Quinn, S. Rogers, B. Shrestha, A. Tripathi, J. J. J. van der Hooft, F. Vargas, K. C. Weldon, M. Witting, H. Yang, Z. Zhang, F. Zubeil, O. Kohlbacher, S. Böcker, T. Alexandrov, N. Bandeira, M. Wang, and P. C. Dorrestein. Feature-based molecular networking in the GNPS analysis environment. *Nat Methods*, 17(9):905–908, 2020.

[258] J. A. November. Digitizing life: the introduction of computers to biology and medicine. PhD thesis. Princeton, USA: Princeton University, 2006.

[259] H. Oberacher, M. Pavlic, K. Libiseller, B. Schubert, M. Sulyok, R. Schuhmacher, E. Csaszar, and H. C. Köfeler. On the inter-instrument and the inter-laboratory transferability of a tandem mass spectral reference library: 2. Optimization and characterization of the search algorithm. *J Mass Spectrom*, 44(4):494–502, 2009.

[260] R. Otter. The number of trees. *Ann Math*, 49(3):583–599, 1948.

[261] M. L. Overton. *Numerical computing with IEEE floating point arithmetic*. Philadelphia: Siam, 2001.

[262] K. G. Owens. Application of correlation analysis techniques to mass spectral data. *Appl Spectrosc Rev*, 27(1):1–49, 1992.

[263] G. Palmisano, D. Antonacci and M. R. Larsen. Glycoproteomic profile in wine: a 'sweet' molecular renaissance. *J Proteome Res*, 9(12):6148–6159, 2010.

[264] D. J. Pappin, P. Hojrup and A. Bleasby. Rapid identification of proteins by peptide-mass fingerprinting. *Curr Biol*, 3(6):327–332, 1993.

[265] C. Y. Park, A. A. Klammer, L. Käll, M. J. MacCoss and W. S. Noble. Rapid and accurate peptide identification from tandem mass spectra. *J Proteome Res*, 7(7):3022–3027, 2008.

[266] W. E. Parkins. The uranium bomb, the calutron, and the space-charge problem. *Phys Today*, 58(5):45–51, 2005.

[267] M. Pavlic, K. Libiseller and H. Oberacher. Combined use of ESI-QqTOF-MS and ESI-QqTOF-MS/MS with mass-spectral library search for qualitative analysis of drugs. *Anal Bioanal Chem*, 386(1):69–82, 2006.

[268] J. E. Peironcely, M. Rojas-Chertó, D. Fichera, T. Reijmers, L. Coulier, J.-L. Faulon and T. Hankemeier. OMG: open molecule generator. *J Cheminformatics*, 4(1):21, 2012.

[269] J. Peng, J. E. Elias, C. C. Thoreen, L. J. Licklider and S. P. Gygi. Evaluation of multidimensional chromatography coupled with tandem mass spectrometry (LC/LC-MS/MS) for large-scale protein analysis: the yeast proteome. *J Proteome Res*, 2(1):43–50, 2003.

[270] D. N. Perkins, D. J. Pappin, D. M. Creasy and J. S. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567, 1999.

[271] R. H. Perry, R. G. Cooks and R. J. Noll. Orbitrap mass spectrometry: instrumentation, ion motion and applications. *Mass Spectrom Rev*, 27(6):661–699, 2008.

[272] J. Pickands III. Statistical inference using extreme order statistics. *Ann Stat*, 3(1):119–131, 1975.

[273] T. Pluskal, T. Uehara and M. Yanagida. Highly accurate chemical formula prediction tool utilizing high-resolution mass spectra, MS/MS fragmentation, heuristic rules, and isotope pattern matching. *Anal Chem*, 84(10):4396–4403, 2012.

[274] G. Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica*, 68(1):145–254, 1937.

[275] R. Prim. Shortest connection networks and some generalizations. *Bell Syst Tech J*, 36(6):1389–1401, 1957.

[276] T. P. Quinn. Stool studies don't pass the sniff test: a systematic review of human gut microbiome research suggests widespread misuse of machine learning. 2021.

[277] E. M. Rains and N. J. Sloane. On Cayley's enumeration of alkanes (or 4-valent trees). *J Integer Seq*, 2(99.1):1, 1999.

[278] R. Raman, S. Raguram, G. Venkataraman, J. C. Paulson and R. Sasisekharan. Glycomics: an integrated systems approach to structure-function relationships of glycans. *Nat Methods*, 2(11):817–824, 2005.

[279] J. L. Ramírez-Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16(1):143–147, 1996.

[280] J. L. Ramírez-Alfonsín. *The Diophantine Frobenius Problem*. Oxford University Press, 2005.

[281] D. F. Ransohoff and A. R. Feinstein. Problems of spectrum and bias in evaluating the efficacy of diagnostic tests. *N Engl J Med*, 299(17):926–930, 1978.

[282] D. F. Ransohoff. Rules of evidence for cancer molecular-marker discovery and validation. *Nat Rev Cancer*, 4(4):309–314, 2004.

[283] D. F. Ransohoff. Bias as a threat to the validity of cancer molecular-marker research. *Nat Rev Cancer*, 5(2):142–149, 2005.

[284] F. Rasche, K. Scheubert, F. Hufsky, T. Zichner, M. Kai, A. Svatoš and S. Böcker. Identifying the unknowns by aligning fragmentation trees. *Anal Chem*, 84(7):3417–3426, 2012.

[285] F. Rasche, A. Svatoš, R. K. Maddula, C. Böttcher and S. Böcker. Computing fragmentation trees from tandem mass spectrometry data. *Anal Chem*, 83(4):1243–1251, 2011.

[286] I. Rauf, F. Rasche, F. Nicolas and S. Böcker. Finding maximum colorful subtrees in practice. *J Comput Biol*, 20(4):311–321, 2013.

[287] G. E. Reid and S. A. McLuckey. 'Top down' protein characterization via tandem mass spectrometry. *J Mass Spectrom*, 37(7):663–675, 2002.

[288] A. Richards, T. Schouwenaars, J. P. How and E. Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *J Guid Control Dyn*, 25(4):755–764, 2002.

[289] L. Ridder, J. J. J. van der Hooft, S. Verhoeven, R. C. H. de Vos, R. J. Bino and J. Vervoort. Automatic chemical structure annotation of an LC-MS(n) based metabolic profile from green tea. *Anal Chem*, 85(12):6033–6040, 2013.

[290] L. Ridder, J. J. J. van der Hooft, S. Verhoeven, R. C. H. de Vos, R. van Schaik and J. Vervoort. Substructure-based annotation of high-resolution multistage MS^n spectral trees. *Rapid Commun Mass Spectrom*, 26(20):2461–2471, 2012.

[291] P. Riley. Three pitfalls to avoid in machine learning. *Nature*, 572(7767):27–29, 2019.

[292] A. L. Rockwood and P. Haimi. Efficient calculation of accurate masses of isotopic peaks. *J Am Soc Mass Spectrom*, 17(3):415–419, 2006.

[293] A. L. Rockwood. Relationship of Fourier transforms to isotope distribution calculations. *Rapid Commun Mass Spectrom*, 9:103–105, 1995.

[294] A. L. Rockwood, M. M. Kushnir and G. J. Nelson. Dissociation of individual isotopic peaks: predicting isotopic distributions of product ions in MS^n. *J Am Soc Mass Spectrom*, 14:311–322, 2003.

[295] A. L. Rockwood, S. L. V. Orden and R. D. Smith. Ultrahigh resolution isotope distribution calculations. *Rapid Commun Mass Spectrom*, 10:54–59, 1996.

[296] A. L. Rockwood and S. L. Van Orden. Ultrahigh-speed calculation of isotope distributions. *Anal Chem*, 68:2027–2030, 1996.

[297] A. L. Rockwood, J. R. Van Orman and D. V. Dearden. Isotopic compositions and accurate masses of single isotopic peaks. *J Am Soc Mass Spectrom*, 15:12–21, 2004.

[298] J. Rodriguez, N. Gupta, R. D. Smith and P. A. Pevzner. Does trypsin cut before proline? *J Proteome Res*, 7(1):300–305, 2008.

[299] P. Roepstorff and J. Fohlman. Proposal for a common nomenclature for sequence ions in mass spectra of peptides. *Biomed Mass Spectrom*, 11(11):601, 1984.

[300] S. Rogers, R. A. Scheltema, M. Girolami and R. Breitling. Probabilistic assignment of formulas to mass peaks in metabolomics experiments. *Bioinformatics*, 25(4):512–518, 2009.

[301] F. Romagné, D. Santesmasses, L. White, G. K. Sarangi, M. Mariotti, R. Hübler, A. Weihmann, G. Parra, V. N. Gladyshev, R. Guigó, and S. Castellano. SelenoDB 2.0: annotation of selenoprotein genes in animals and their genetic diversity in humans. *Nucleic Acids Res*, 42(Database issue):D437–D443, 2014.

[302] L. Ruddigkeit, R. van Deursen, L. C. Blum and J.-L. Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *J Chem Inf Model*, 52:2864–2875, 2012.

[303] C. Ruttkies, E. L. Schymanski, S. Wolf, J. Hollender and S. Neumann. MetFrag relaunched: incorporating strategies beyond in silico fragmentation. *J Cheminformatics*, 8:3, 2016.

[304] R. G. Sadygov and J. R. Yates III. A hypergeometric probability model for protein identification and validation using tandem mass spectral data and protein sequence databases. *Anal Chem*, 75(15):3792–3798, 2003.

[305] T. Sakurai, T. Matsuo, H. Matsuda and I. Katakuse. PAAS 3: a computer program to determine probable sequence of peptides from mass spectrometric data. *Biomed Mass Spectrom*, 11(8):396–399, 1984.

[306] A. Salomaa. Counting (scattered) subwords. *B Euro Assoc Theo Comp Sci*, 81:165–179, 2003.

[307] F. Sanger, S. Nicklen and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A*, 74(12):5463–5467, 1977.

[308] R. A. Sayle. So you think you understand tautomerism? *J Comput-Aided Mol Des*, 24(6-7):485–496, 2010.

[309] K. Scheubert, F. Hufsky and S. Böcker. Computational mass spectrometry for small molecules. *J Cheminformatics*, 5:12, 2013.

[310] K. Scheubert, F. Hufsky and S. Böcker. Multiple mass spectrometry fragmentation trees revisited: boosting performance and quality. In: *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2014)*. Vol. 8701. Lect Notes Comput Sci. Springer, Berlin, 2014, pages 217–231.

[311] K. Scheubert, F. Hufsky, D. Petras, M. Wang, L.-F. Nothias, K. Dührkop, N. Bandeira, P. C. Dorrestein, and S. Böcker. Significance estimation for large scale metabolomics annotations by spectral matching. *Nat Commun*, 8:1494, 2017.

[312] K. Scheubert, F. Hufsky, F. Rasche and S. Böcker. Computing fragmentation trees from metabolite multiple mass spectrometry data. In: *Proc. of Research in Computational Molecular Biology (RECOMB 2011)*. Vol. 6577. Lect Notes Comput Sci. Springer, Berlin, 2011, pages 377–391.

[313] P. v. R. Schleyer, C. Maerker, A. Dransfeld, H. Jiao and N. J. R. van Eikema Hommes. Nucleus-independent chemical shifts: a simple and efficient aromaticity probe. *Journal of the American Chemical Society*, 118(26):6317–6318, 1996.

[314] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Series in Discrete Mathematics & Optimization. Wiley, 1998.

[315] E. L. Schymanski and S. Neumann. The critical assessment of small molecule identification (CASMI): Challenges and solutions. *Metabolites*, 3(3):517–538, 2013.

[316] E. L. Schymanski, C. M. J. Gallampois, M. Krauss, M. Meringer, S. Neumann, T. Schulze, S. Wolf, and W. Brack. Consensus structure elucidation combining GC/EI-MS, structure generation and calculated properties. *Anal Chem*, 84(7):3287–3295, 2012.

[317] E. L. Schymanski, C. Ruttkies, M. Krauss, C. Brouard, T. Kind, K. Dührkop, F. R. Allen, A. Vaniya, D. Verdegem, S. Böcker, J. Rousu, H. Shen, H. Tsugawa, T. Sajed, O. Fiehn, B. Ghesquière, and S. Neumann. Critical Assessment of Small Molecule Identification 2016: automated methods. *J Cheminformatics*, 9:22, 2017.

[318] J. Seidler, N. Zinn, M. E. Boehm and W. D. Lehmann. De novo sequencing of peptides by MS/MS. *Proteomics*, 10(4):634–649, 2010.

[319] J. Senior. Partitions and their representative graphs. *Amer J Math*, 73(3):663–689, 1951.

[320] M. W. Senko, S. C. Beu and F. W. McLaffertycor. Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions. *J Am Soc Mass Spectrom*, 6(4):229–233, 1995.

[321] O. Serang and W. Noble. A review of statistical methods for protein identification using tandem mass spectrometry. *Stat Interface*, 5(1):3–20, 2012.

[322] B. Shan, B. Ma, K. Zhang and G. Lajoie. Complexities and algorithms for glycan sequencing using tandem mass spectrometry. *J Bioinform Comput Biol*, 6(1):77–91, 2008.

[323] H. Shen, K. Dührkop, S. Böcker and J. Rousu. Metabolite identification through multiple kernel learning on fragmentation trees. *Bioinformatics*, 30(12):i157–i164, 2014. Proc. of *Intelligent Systems for Molecular Biology* (ISMB 2014).

[324] X. Shen, R. Wang, X. Xiong, Y. Yin, Y. Cai, Z. Ma, N. Liu, and Z.-J. Zhu. Metabolic reaction network-based recursive metabolite annotation for untargeted metabolomics. *Nat Commun*, 10(1):1516, 2019.

[325] Q. Sheng, Y. Mechref, Y. Li, M. V. Novotny and H. Tang. A computational approach to characterizing bond linkages of glycan isomers using matrix-assisted laser desorption/ionization tandem time-of-flight mass spectrometry. *Rapid Commun Mass Spectrom*, 22(22):3561–3569, 2008.

[326] H. Shin, M. P. Sampat, J. M. Koomen and M. K. Markey. Wavelet-based adaptive denoising and baseline correction for MALDI TOF MS. *OMICS*, 14(3):283–295, 2010.

[327] F. Sikora. An (almost complete) state of the art around the graph motif problem. Tech. rep. Available from `http://www-igm.univ-mlv.fr/~fsikora/pub/GraphMotif-Resume.pdf`. France, 2010.

[328] R. M. Silverstein, F. X. Webster and D. Kiemle. *Spectrometric Identification of Organic Compounds*. 7th. Wiley, 2005, page 512.

[329] G. Siuzdak. *The Expanding Role of Mass Spectrometry in Biotechnology*. second. MCC Press, 2006, page 257.

[330] D. H. Smith, N. A. Gray, J. G. Nourse and C. W. Crandell. The DENDRAL project: recent advances in computer-assisted structure elucidation. *Anal Chim Acta*, 133(4):471–497, 1981.

[331] R. K. Snider. Efficient calculation of exact mass isotopic distributions. *J Am Soc Mass Spectrom*, 18(8):1511–1515, 2007.

[332] K. C. Sou, J. Weimer, H. Sandberg and J. Karl Henrik. Scheduling smart home appliances using mixed integer linear programming. In: *Proc. of IEEE Conference on Decision and Control and European Control Conference (CDC 2011)*. IEEE. 2011.

[333] M. Spivak, J. Weston, L. Bottou, L. Käll and W. S. Noble. Improvements to the Percolator algorithm for peptide identification from shotgun proteomics data sets. *J Proteome Res*, 8(7):3737–3745, 2009.

[334] H. Steen and M. Mann. The ABC's (and XYZ's) of peptide sequencing. *Nat Rev Mol Cell Biol*, 5:699–711, 2004.

[335] S. E. Stein. Chemical substructure identification by mass spectral library searching. *J Am Soc Mass Spectrom*, 6(8):644–655, 1995.

[336] S. E. Stein and D. R. Scott. Optimization and testing of mass spectral library search algorithms for compound identification. *J Am Soc Mass Spectrom*, 5(9):859–866, 1994.

[337] F. M. Steiner, B. C. Schlick-Steiner, A. Nikiforov, R. Kalb and R. Mistrik. Cuticular hydrocarbons of Tetramorium ants from central Europe: analysis of GC-MS data with self-organizing maps (SOM) and implications for systematics. *J Chem Ecol*, 28(12):2569–2584, 2002.

[338] A. Sticker, L. Martens and L. Clement. Mass spectrometrists should search for all peptides, but assess only the ones they care about. *Nat Methods*, 14(7):643–644, 2017.

[339] J. D. Storey, J. M. Akey and L. Kruglyak. Multiple locus linkage analysis of genomewide expression in yeast. *PLoS Biol*, 3(8):e267, 2005.

[340] J. J. Sylvester and W. J. Curran Sharp. Problem 7382. *Educational Times*, 37:26, 1884.

[341] D. L. Tabb, A. Saraf and J. R. Yates. GutenTag: high-throughput sequence tagging via an empirically derived fragmentation model. *Anal Chem*, 75(23):6415–6421, 2003.

[342] H. Tang, Y. Mechref and M. V. Novotny. Automated interpretation of MS/MS spectra of oligosaccharides. *Bioinformatics*, 21 Suppl 1:i431–i439, 2005. Proc. of *Intelligent Systems for Molecular Biology* (ISMB 2005).

[343] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J Comput System Sci*, 18(2):110–127, 1979.

[344] J. A. Taylor and R. S. Johnson. Sequence database searches via de novo peptide sequencing by tandem mass spectrometry. *Rapid Commun Mass Spectrom*, 11(9):1067–1075, 1997.

[345] M. The and L. Käll. MaRaCluster: a fragment rarity metric for clustering fragment spectra in shotgun proteomics. *J Proteome Res*, 15(3):713–720, 2016.

[346] W. Timm, A. Scherbart, S. Böcker, O. Kohlbacher and T. W. Nattkemper. Peak intensity prediction in MALDI-TOF mass spectrometry: a machine learning study to support quantitative proteomics. *BMC Bioinformatics*, 9:443, 2008.

[347] Y. S. Ting, J. D. Egertson, S. H. Payne, S. Kim, B. MacLean, L. Käll, R. Aebersold, R. D. Smith, W. S. Noble, and M. J. MacCoss. Peptide-centric proteome analysis: an alternative strategy for the analysis of tandem mass spectrometry data. *Mol Cell Proteomics*, 14(9):2301–2307, 2015.

[348] S. Tiwary, R. Levy, P. Gutenbrunner, F. Salinas Soto, K. K. Palaniappan, L. Deming, M. Berndl, A. Brant, P. Cimermancic, and J. Cox. High-quality ms/ms spectrum prediction for data-dependent and data-independent acquisition data analysis. *Nat Methods*, 16(6):519–525, 2019.

[349] N. H. Tran, X. Zhang, L. Xin, B. Shan and M. Li. De novo peptide sequencing by deep learning. *Proc Natl Acad Sci U S A*, 114(31):8247–8252, 2017.

[350] C.-C. Tsou, D. Avtonomov, B. Larsen, M. Tucholska, H. Choi, A.-C. Gingras and A. I. Nesvizhskii. DIA-Umpire: comprehensive computational framework for data-independent acquisition proteomics. *Nat Methods*, 12(3):258–264, 2015.

[351] J. Urban, C. Jin, K. A. Thomsson, N. G. Karlsson, C. M. Ives, E. Fadda and D. Bojar. Predicting glycan structure from tandem mass spectrometry via deep learning. *Nat Methods*, 21(7):1206–1215, 2024.

[352] J. van Lint and R. Wilson. *A Course in Combinatorics*. Cambridge University Press, 2001.

[353] A. Varki, R. D. Cummings, J. D. Esko, P. Stanley, G. W. Hart, M. Aebi, D. Mohnen, T. Kinoshita, N. H. Packer, J. H. Prestegard, R. L. Schnaar, and P. H. Seeberger, eds. *Essentials of Glycobiology*. fourth. Cold Spring Harbor (NY): Cold Spring Harbor Laboratory Press, 2022.

[354] K. Varmuza and W. Werther. Mass spectral classifiers for supporting systematic structure elucidation. *J Chem Inf Comput Sci*, 36(2):323–333, 1996.

[355] J. D. Velasco. The prior probabilities of phylogenetic trees. *Biol Philos*, 23(4):455–473, 2008.

[356] R. Venkataraghavan, F. W. McLafferty and G. E. van Lear. Computer-aided interpretation of mass spectra. *Org Mass Spectrom*, 2(1):1–15, 1969.

[357] F. Wang, J. Liigand, S. Tian, D. Arndt, R. Greiner and D. S. Wishart. CFM-ID 4.0: more accurate ESI-MS/MS spectral prediction and compound identification. *Anal Chem*, 93(34):11692–11700, 2021.

[358] J. Wang, W. Wang, P. A. Kollmann and D. A. Case. Automatic atom type and bond type perception in molecular mechanical calculations. *J Mol Graph Model*, 25:247–260, 2006.

[359] M. Wang, G. Audi, F. Kondev, W. Huang, S. Naimi and X. Xu. The AME2016 atomic mass evaluation (II). Tables, graphs and references. *Chinese Physics C*, 41(3):030003, 2017.

[360] M. Wang, W. Huang, F. Kondev, G. Audi and S. Naimi. The AME 2020 atomic mass evaluation (II). tables, graphs and references. *Chinese Physics C*, 45(3):030003, 2021.

[361] M. Wang *et al.* Sharing and community curation of mass spectrometry data with Global Natural Products Social molecular networking. *Nat Biotechnol*, 34(8):828–837, 2016.

[362] M. S. Waterman and M. Vingron. Rapid and accurate estimates of statistical significance for sequence data base searches. *Proc Natl Acad Sci U S A*, 91(11):4625–4628, 1994.

[363] J. Watrous, P. Roach, T. Alexandrov, B. S. Heath, J. Y. Yang, R. D. Kersten, M. van der Voort, K. Pogliano, H. Gross, J. M. Raaijmakers, B. S. Moore, J. Laskin, N. Bandeira, and P. C. Dorrestein. Mass spectral molecular networking of living microbial colonies. *Proc Natl Acad Sci U S A*, 109(26):E1743–E1752, 2012.

[364] J. T. Watson and O. D. Sparkman. *Introduction to Mass Spectrometry: Instrumentation, Applications, and Strategies for Data Interpretation*. Wiley, 2007.

[365] W. Werther, H. Lohninger, F. Stancl and K. Varmuza. Classification of mass spectra: a comparison of yes/no classification methods for the recognition of simple structural properties. *Chemom Intell Lab Syst*, 22(1):63–76, 1994.

[366] T. Wieland. Konstruktionsalgorithmen bei molekularen Graphen und deren Anwendung. *MATCH Commun Math Comput Chem*, 39:7–155, 1997.

[367] H. Wilf. *generatingfunctionology*. second. Freely available from `http://www.math.upenn.edu/~wilf/DownldGF.html`. Academic Press, 1994.

[368] S. Wolf, S. Schmidt, M. Müller-Hannemann and S. Neumann. In silico fragmentation for computer assisted identification of metabolite mass spectra. *BMC Bioinformatics*, 11:148, 2010.

[369] Y. Wu, Y. Mechref, I. Klouckova, M. V. Novotny and H. Tang. A computational approach for the identification of site-specific protein glycosylations through ion-trap mass spectrometry. In: *Proc. of RECOMB 2006 satellite workshop on Systems Biology and Computational Proteomics (RECOMB 2006)*. Vol. 4532. Lect Notes Comput Sci. San Diego, CA, USA: Springer, Berlin, 2007, pages 96–107.

[370] J. Yates, P. Griffin, L. Hood and J. Zhou. Computer aided interpretation of low energy MS/MS mass spectra of peptides. In: *Techniques in Protein Chemistry II*. Ed. by J. Villafranca. San Diego: Academic Press, 1991, pages 477–485.

[371] A. L. Yergey and A. K. Yergey. Preparative scale mass spectrometry: a brief history of the calutron. *J Am Soc Mass Spectrom*, 8(9):943–953, 1997.

[372] J. A. Yergey. A general approach to calculating isotopic distributions for mass spectrometry. *Int J Mass Spectrom Ion Phys*, 52(2–3):337–349, 1983.

[373] H. Yoshida, R. Leardi, K. Funatsu and K. Varmuza. Feature selection by genetic algorithms for mass spectral classifiers. *Anal Chim Acta*, 446(1-2):483–492, 2001.

[374] A. Young, H. Röst and B. Wang. Tandem mass spectrum prediction for small molecules using graph transformers. *Nat Mach Intell*, 6(4):404–416, 2024.

[375] F. Yu, G. C. Teo, A. T. Kong, K. Fröhlich, G. X. Li, V. Demichev and A. I. Nesvizhskii. Analysis of DIA proteomics data using MSFragger-DIA and FragPipe computational platform. *Nat Commun*, 14:4154, 2023.

[376] J. Zaia. Mass spectrometry of oligosaccharides. *Mass Spectrom Rev*, 23(3):161–227, 2004.

[377] Z. Zhang. Prediction of low-energy collision-induced dissociation spectra of peptides. *Anal Chem*, 76(14):3908–3922, 2004.

[378] X.-X. Zhou, W.-F. Zeng, H. Chi, C. Luo, C. Liu, J. Zhan, S.-M. He, and Z. Zhang. pDeep: predicting MS/MS spectra of peptides with deep learning. *Anal Chem*, 89(23):12690–12697, 2017.

[379] R. Zubarev and M. Mann. On the proper use of mass accuracy in proteomics. *Mol Cell Proteomics*, 6(3):377–381, 2007.

Glossary

Acronym	Description (page or section)
CID	Collision-Induced Dissociation (Sec. 1.5)
DAG	directed acyclic graph
DP	Dynamic Programming (Sec. 15.5)
EI	Electron Ionization, formerly known as Electron Impact ionization (p. 9)
EM	Expectation Maximization (Sec. 6.6)
ESI	Electrospray Ionization (p. 9)
ETD	Electron Transfer Dissociation (Sec. 1.5)
FDR	False Discovery Rate (Sec. 6.1)
FT-ICR	Fourier Transform Ion Cyclotron Resonance
GC	Gas Chromatography (Sec. 1.6.2)
ILP	Integer Linear Program(ming) (Sec. 9.4)
LC	Liquid Chromatography (Sec. 1.6.2)
LP	Linear Program(ming) (Sec. 9.4)
LSTM	Long Short-Term Memory (neural network)
MALDI	Matrix-Assisted Laser Desorption/Ionization (p. 9)
MS	mass spectrometry
MS/MS	tandem mass spectrometry (Sec. 1.5)
MS^n	multistage mass spectrometry
MSA	Multiple Sequence Alignment
NMR	Nuclear Magnetic Resonance
PEP	Posterior Error Probability (Sec. 6.6)
PSM	Peptide-Spectrum Match, plural PSMs
PTAS	Polynomial-Time Approximation Scheme (Sec. 15.7)
PTM	Post-Translational Modification, plural PTMs
RDBE	Ring Double Bond Equivalent (Sec. 8.4)
SVM	Support Vector Machine
TDC	Target-Decoy Competition (Sec. 6.3)
TDSS	Target-Decoy Separate Search (Sec. 6.5)
TOF	Time of Flight (p. 10)