2 Peptide *De Novo* Sequencing

[THIS CHAPTER IS "FEATURE COMPLETE" AND READY FOR PROOFREADING!]

> "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 mass spectra since at least the 1960's: For example, Biemann, Cone, Webster, and Arsenault [20] 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 [39] 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 [48] at the conference for Research in Computational Molecular Biology. Some might argue that this history already started back in 1997, when Taylor and Johnson [226] 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 [64], 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 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 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 feasible argument, but it is wrong: We just mention a few counter-arguments. In Eukaryotes, a single gene can correspond to ten thousands of proteins due to alternative splicing. Most proteins are edited and modified after translation, and there exists a huge variety of Posttranslational Modifications for this purpose. Certain proteins are not even encoded in the genome, for example the antibiotic Actinomycin D. Not every species is sequenced, and not every gene is annotated. Proteogenomics, an emerging scientific field at the intersection of proteomics and genomics, uses proteomic information from mass spectrometry to improve gene annotations; see Sec. 16.1. This list is most likely incomplete, but it is sufficient to make the point.

[1] Even others might argue that this history started with DENDRAL back in 1965 [144, 145], see Sec. 13.6; but I would object, as DENDRAL projects did not care about specification, generalizability, correctness, or running time of the developed algorithms.

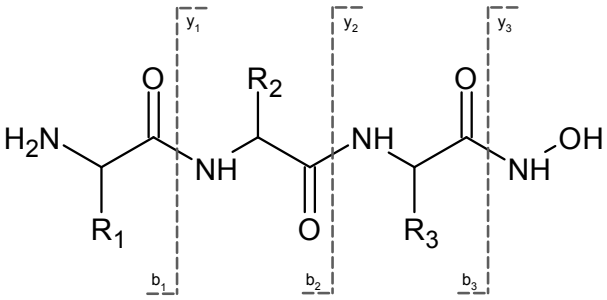

Figure 2.1: Fragmentation of a peptide into b and y ions. **[TODO: GRAPHICS BUGGY.]**

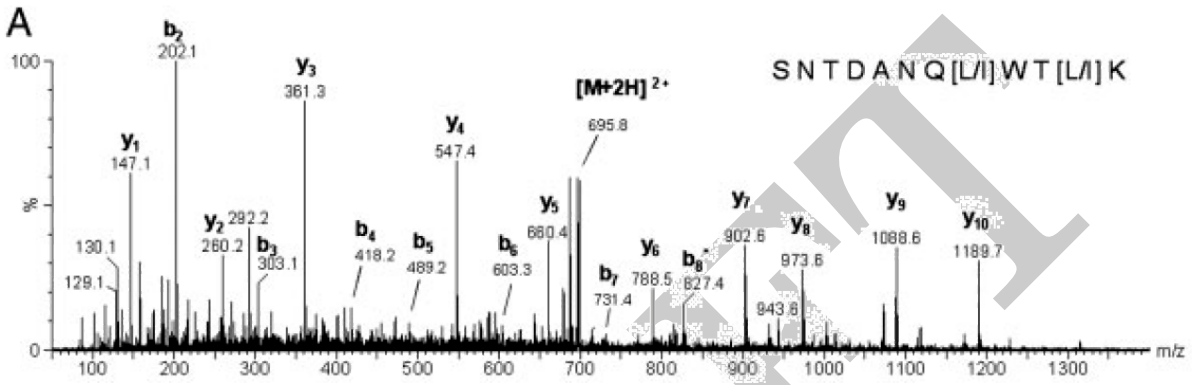

Figure 2.2: Expert-annotated tandem MS spectrum of the peptide ???. **[TODO: ASKED WOLF LEHMANN FOR A SPECTRUM FROM SEIDLER ET AL, PROTEOMICS 2010]**

And why are we sequencing peptides and not proteins? For the moment, a simple answer should be sufficient: If we *could* sequence proteins, then we *would* sequence proteins; but we *cannot*. More details can be found in Chapter 11.

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 noble gas atoms, causing them to fragment by 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 has been 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. Only after finding an algorithmic solution for the simple problem, 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 *length* of s. The unique string of length zero is called *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 *substring* of s. Deliberately, we did not excluded 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 from. 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. 2.6. 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 true 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 these masses in application? 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.) To this end, 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

$$C_5H_7N_1O_3 + C_3H_5N_1O_2 + C_6H_{11}N_1O_1 + H_2O = C_{14}H_{25}N_3O_7$$

symb.	TLC	amino acid	molecular formula	mass (Da)
A	Ala	alanine	$C_3H_5N_1O_1$	71.037114
C	Cys	cysteine	$C_3H_5N_1O_1S_1$	103.009184
D	Asp	aspartic acid	$C_4H_5N_1O_3$	115.026943
E	Glu	glutamic acid	$C_5H_7N_1O_3$	129.042593
F	Phe	phenylalanine	$C_9H_9N_1O_1$	147.068414
G	Gly	glycine	$C_2H_3N_1O_1$	57.021464
H	His	histidine	$C_6H_7N_3O_1$	137.058912
I	Ile	isoleucine	$C_6H_{11}N_1O_1$	113.084064
K	Lys	lysine	$C_6H_{12}N_2O_1$	128.094963
L	Leu	leucine	$C_6H_{11}N_1O_1$	113.084064
M	Met	methionine	$C_5H_9N_1O_1S_1$	131.040485
N	Asn	asparagine	$C_4H_6N_2O_2$	114.042927
P	Pro	proline	$C_5H_7N_1O_1$	97.052764
Q	Gln	glutamine	$C_5H_8N_2O_2$	128.058578
R	Arg	arginine	$C_6H_{12}N_4O_1$	156.101111
S	Ser	serine	$C_3H_5N_1O_2$	87.032028
T	Thr	threonine	$C_4H_7N_1O_2$	101.047678
V	Val	valine	$C_5H_9N_1O_1$	99.068414
W	Trp	tryptophan	$C_{11}H_{10}N_2O_1$	186.079313
Y	Tyr	tyrosine	$C_9H_9N_1O_2$	163.063329

Table 2.1: Proteogenic amino acids with symbol, 3-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 small mass difference of only 0.036385 Da. **[ToDo: Check masses]**

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 9.

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*

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

or *peaks*, depending on the context. The *parent mass* M of a string s is simply its mass, $M = \mu(s)$. We may assume that we know the parent 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 "unhandy", 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 how 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".

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 assumptions 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. 2.5.5.

5. The MS instrument records exact masses of peptide fragments. In reality, measured masses will deviate from these theoretical and exact masses. This will be covered in Sec. 2.5.6, but we will come back to this problem repeatedly throughout this book.

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 of our aspiration for generalizability. Only when we come to the optimization version of our algorithm, we can 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.

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 application. 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 Sec. 17.2.

A *directed graph* $G = (V, E)$ consists of a set of vertices 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

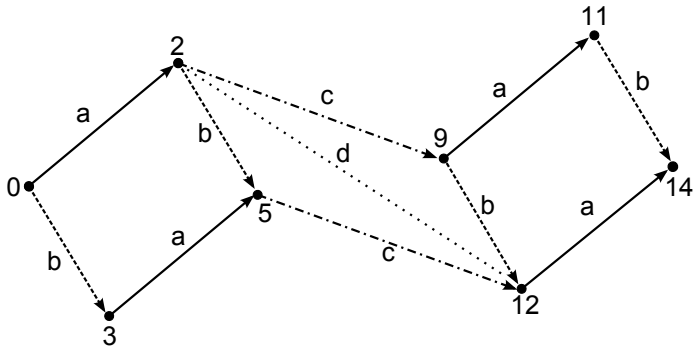

Figure 2.4: Spectrum graph for Example 2.1. Bold edges show the valid path corresponding to string $s = \mathrm{acab}$. **[TODO: NO EDGE STYLES, BOLD EDGES FOR TRUE PATH.]**

$p = u_0 u_1 \dots u_l$ of vertices of G, such that $u_{i-1}u_i$ is an edge of G for all $i = 1,\dots,l$. 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 vertex $v \in V$. A directed graph is *acyclic* if it does not contain any cycles. Informally speaking, "acyclic" means that we cannot walk away from some vertex 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 vertex 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 vertex u, all edges leaving u are labeled differently. It is easy to check that the spectrum graph is acyclic; in fact, its vertices are ordered, and an edge from vertex 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 parent mass of the unknown string. Vertex 0 is a *source* of the graph as it has not incoming edges; vertex 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 $\mathcal{M} = \{x_0, x_1, \dots, x_{n-1}, x_n, y_n, y_{n-1}, \dots, y_1, y_0\}$ with $x_0 = 0$, $y_0 = M$, and

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

By Observation 2, we infer that $x_j + y_j = M$ holds for all $j = 1,\dots,n$; 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 parent 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 $v_0 v_1 \dots v_n$ through the spectrum graph $G(\mathcal{M})$ from 0 to M: From vertex v_{j-1} we follow the edge labeled s_j to v_j, for all $j = 1,\dots,n$.

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 vertices 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 observation: 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 that a path in the spectrum graph $G=(V,E)$ is *valid* if it starts in 0 and ends in M, and for $V=\{x_0,\dots,x_n,y_n,\dots,y_0\}$ from (2.2), the path visits *exactly one* of the two vertices x_i,y_i for every $i=1,\dots,n$.

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 $p=v_0v_1\dots v_{n+1}$ 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 $n-j$ has mass $M-v_j$, for all $i=0,\dots,n$. 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. 17.5. This 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 find such an efficient algorithm, that 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 vertices, and we test whether the induced graph contains a path from 0 to M through all vertices, what can be done in linear time. If so, then the resulting string s satisfies $\mathcal{M}(s)=\mathcal{M}$ and we are done. Running time of this algorithm is $O(2^n\cdot n)$ and, hence, the algorithm is limited to rather

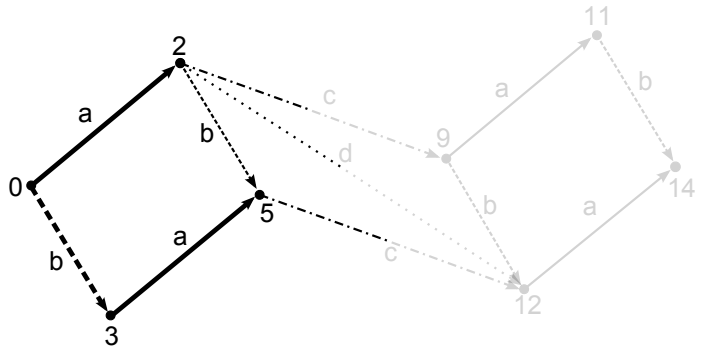

Figure 2.5: Prefix path $x_0 = 0, x_1 = 2$ and suffix path $x_0 = 0, x_2 = 3, x_3 = 5$ for Example 2.1. These paths form a valid pair.

small strings. In practice, running time 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, compare to 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 $V = \mathcal{M} = \{x_0,\dots,x_n,y_n,\dots,y_0\}$ 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 vertices is part of the path. To this end, we use a detour that is conceptually slightly simpler: We ignore $y_n,\dots,y_0$ and concentrate on vertices $x_0,\dots,x_n$. 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 vertex-disjoint, with the exception of the start vertex $x_0 = 0$. Furthermore, each vertex $x_1,\dots,x_n$ 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 are *valid pair* if, for all $l = 1,\dots,\max\{i,j\}$, vertex 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 vertices from $x_0,\dots,x_n$, and let p_2 be the remaining part of the path with vertices from $y_n,\dots,y_0$. Now, we can *flip* $p_2 = v_0 \dots v_l$ by setting $p_2^* := u_l \dots u_0$ with $u_j := M - v_j$ for all $j = 1,\dots,l$. 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 vertices 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 vertex of the prefix path, and that y_j is the first vertex 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 vertex of p, for all $l = 1,\ldots,\max\{i,j\}$. But we also have to make sure that all vertices 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.17.3. We define a binary matrix $D[0\ldots n, 0\ldots n]$ 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.[2] Clearly, $D[0,0] = 1$ holds, as well as $D[j,j] = 0$ for $j = 1,\ldots,n$. 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 $\mathscr{M} := \mathscr{M}(s) = \{0,2,3,5,9,11,12,14\}$ 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_0x_2, and the suffix path is $x_0x_1x_3$. 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 vertex 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 vertices x_l for $l = 1,\ldots,i-2$ 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 $l \in \{1,\ldots,i-2\}$. 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 later is true, by definition of D, if and only if $D[l,j] = 1$. So, for all $l = 1,\ldots,i-2$, 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.

[2] 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 **[TODO: DRAW FIGURE, SEE LECTURE NOTES]**.

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

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

We initialize $D[0,0] = 1$ and $D[j,j] = 0$ for all $j \geq 1$. In (2.3) we assume that $\max \emptyset = 0$, to simplify the formalism. Note that taking the maximum is only some "math voodoo", that allows us to write up the equation more easily: The expression gets 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 Alg. 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 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 Alg. 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, that 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

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 $i = 0,\ldots,M$
  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 $D[0,0] \leftarrow 1$ and $D[i,i] \leftarrow 0$ for $i = 1,\ldots,n$
  for $i \leftarrow 0,\ldots,n$ **do** ▷ Fill the matrix
    for $j \leftarrow 0,\ldots,n$ **do**
      if $i \neq j$ **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.

$j' < j$, 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 Alg. 2.2 for the pseudocode of this algorithm.

2.5 Getting rid of the unrealistic assumptions

Hitherto, we have used some unrealistic assumptions to simplify the problem, that 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 book. After all, this is only the beginning of our journey through the realms of computational mass spectrometry. These issues include:

function BACKTRACINGIDEALDATA(matrix $D[0\ldots n, 0\ldots n]$, integers i, j, graph $G = (V, E)$)
  Let $\{x_0, \ldots, x_n, y_n, \ldots, y_0\} := V$ 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.

- The general problem of matching mass spectra; this will be covered in Sec. 4.1.
- Penalizing (or rather not penalizing) additional peaks in Sec. 2.5.1; this will be covered in more detail in Sec. 4.3.
- Penalizing missing peaks in Sec. 2.5.3; we will come back to this in Sec. 4.1.
- "Strings without order" in Sec. 2.5.3; this leads to the definition of "compomers" in the next chapter.
- A sensible way of dealing with mass inaccuracies, see Sec. 2.5.6. In Sec. 4.2, we will introduce a statistical model for mass inaccuracies.

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 application, we will not only record the masses of prefixes and suffixes of our peptide string but, in addition, 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 $\mathcal{M} = \{x_0, \ldots, x_n, y_n, \ldots, y_0\}$ 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 book, 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 $Q[0\ldots n, 0\ldots n]$ 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. 4.3. We initialize $Q[0,0] = 0$ and $Q[j,j] = -\infty$ for $j = 1,\ldots,n$. 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? Different from ideal data, the optimal valid path may skip the vertex pair x_n, y_n altogether. To this end, 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 (2.5). 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:

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

At this point, our scoring function w serves a single purpose: Entries $Q[l,j]$ and $Q[i,l]$ are not taking 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 application.

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 suffix 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 $Q[i, j] \leq \max_{l=0,\ldots,i-1}\{Q[l, j] + w(x_l, x_i)\}$, and it remains to be shown that $Q[L, j] = \max_{l=0,\ldots,i-1}\{Q[l, j]\}$. 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 $O(n)$, compare to Exercise 2.6. From the theoretical standpoint, this is a huge increase in running time; luckily, n is rather small in application with $n \leq 100$ in most cases, so computation time will hardly ever reach one second on a moder 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 $D[i', j']$ where $D[i, j]$ in the maxima of (2.5) has progressed from, so $D[i, j] = D[i', j'] + 1$. 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 $p = u_0 u_1 \ldots u_l$ 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 $\mathcal{M} = \{x_0, \ldots, x_n, y_n, \ldots, y_0\}$ with $x_i + y_i = M$ for all $i = 0, \ldots, n$. 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 $\max\{Q[i, j] + w(x_i, y_j) : x_i y_i \in E\}$ 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 y_j that form a valid pair.

Looking at the lemma, you will notice one important change: The maximum weight of a valid path is $\max\{Q[i, j] + w(x_i, y_j) : x_i y_i \in E\}$ whereas previously, we searched for $\max\{Q[i, j] : x_i y_i \in E\}$. 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 $x_i y_i$ 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 vertices, not edges! The conceptually most elegant way to get around this dilemma, is to push the weight of a vertex to all of its incoming edges. In this way, weight $w(u, v)$ corresponds to the weight of vertex v. As at most one of the edges entering v is part of the path, we add the weight of a vertex if and only if the path passes through that vertex.

This is different from our initial definition of the matrix Q, because now, a suffix path from 0 to x_j does not longer explain the peak y_j: The first edge of the flipped suffix path is $y_j y$ for some vertex $y > y_j$, and $w(y_j, y)$ tells us something about the vertex (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 $\max\{Q[i, j] + w(x_i, y_j) : x_i y_i \in E\}$ 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

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, $\max\{Q[i,j]+w(x_i,y_j) : x_iy_i \in E\}$ is the maximum number of peaks that can be explained by any string, ignoring mass 0.

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 application, 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', that 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 less peaks, so we force the approach to use these larger steps as prudent 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 $y \notin \mathscr{M}$ and $M-y \notin \mathscr{M}$. Then, a "gap string" $z = z_1z_2$, 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 2.13. In Sec. 4.1, 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 $\mathscr{M}$, plus the number of proper suffix masses of s present in $\mathscr{M}$. This score is easy to incorporate into branch-and-bound approaches, as it allows us to truncate the search space.

Consider the "true" string $s = \mathtt{aaabb}$ for the weighted alphabet from Example 2.1. Let $\mathscr{M} := \mathscr{M}(s) = \{0,2,3,4,6,8,9,10,12\}$ 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 $\mathscr{M}$, leading to a score of 8. Where is the problem? Consider the string $s' = \mathtt{aaaaaa}$: This string has five proper prefixes and five proper suffixes, all of which are present in $\mathscr{M}$, 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 $\mathscr{M}$. In fact, the string s' explains only seven out of nine peaks in $\mathscr{M}$. So, we should be able to tell that this explanation is worse, without having to rely on scoring missing peaks.

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 application, see Exercises 3.12 and 3.13. We will now show how 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 $w_A \equiv w$ with $w(x,y) \in \{1, -\infty\}$ from (2.4). We will deliberately *not* used a general scoring function in our presentation; we will come back 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 $\mathscr{M} \setminus \{0,M\}$ explained by any prefix path x_0 to x_i and suffix path x_0 to x_j. Here, vertices that are present in both the prefix path and the suffix path are counted only once; but we do 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 re-use recurrence (2.5) for Q', but extend it by the calculation of the diagonal matrix elements:

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

Once more, recall that $w_A(x,y) = 1$ for $xy \in E$, and $w_A(x,y) = -\infty$ 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 2.14 for a proof, and see Alg. 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 $Q'[i,j] \neq -\infty$, what implies that there is a prefix path to x_i and a suffix path to y_j. These paths do not have to form a valid pair; but at least, they exist. For $i \neq j$, the argumentation 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 $Q'[i,L'] = \max_{l=0,\ldots,j-1}\{Q'[i,l] : y_j y_l \in E\}$ what 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 vertex to all incoming edges, brakes 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 vertices (peaks), not edges; compare to Exercise 2.12. To this end, we can define the weight of a path to be the sum of vertex weights; and, we can define the *valid* weight of a path as the sum of weight where, if x_i and y_i are present simultaneously, only the larger weight is added to the weight of the path. See Exercise 2.16 for details.

function PEPTIDESEQUENCING(set of masses $\mathcal{M}$, parent 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 $maxscore \leftarrow -\infty$ ▷ 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 $Q'[i,j] > maxscore$ **then**
        Let $maxscore \leftarrow Q'[i,j]$ and $(i',j') \leftarrow (i,j)$
      end if
    end for
  end for
  Return (i',j') with score $maxscore$
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.

2.5.5 The b and y ion series

So far, we have assumed that the mass of a prefix or suffix fragment is simply the sum of masses of the constituting amino acid residues. The reality is slightly more complicated: There exist (at least) six different ion series, three for prefix fragments and three for suffix fragments, each with its own mass modification. In this section, we will cover only the two most prominent ion series of peptide fragmentation by Collision Induced Dissociation (CID); recall that this is the most widely used fragmentation technique for peptide sequencing. Other ion series will be addressed in Sec. 4.4 below. Unfortunately, it turns out there is no straightforward way to generalize the approach presented in this chapter to more than two ion series, see Chapter 8.

The two ion series that we want to consider here, are *b ions* for prefix fragments, and *y ions* for suffix fragments. First, we want to take into account that peptide fragmentation is usually carried out with protonated ions, where both prefix and suffix fragments carry an additional proton. Take a look at Fig. 2.1: You can see that the mass of the prefix (b ion) is modified by $+H^+$ (+1.007276 Da), whereas the mass of the suffix (y ion) is by modified by $+H_2OH^+$ (+19.017841 Da). But different from what Fig. 2.1 might suggest, peptide modification must not be thought of as "cutting the peptide molecule into two parts"; we will see in Sec. 4.4 that this intuition is incorrect.

Taking into account the b and y ion series, does not require any changes to our approach. Regarding symmetry, we only made use of the fact that for any peak with mass $m \in \mathcal{M}$ there is also a peak with mass $M - m \in \mathcal{M}$ for parent mass M. Assume that $\mathcal{M}^*$ are the masses recorded by the MS instrument for parent mass M^*. We define

$$\mathcal{M} := \mathcal{M}^* - 1.007276 := \{m - 1.007276 : m \in \mathcal{M}^*\}$$

and, henceforth, we may assume that prefix have ideal masses, whereas suffixes are modified by $+H_2O$ (+18.010565 Da). Finally, we define $M := M^* - 1.007276$ and, again, we may assume that for ideal data, each prefix mass x_i has a complementing suffix mass y_i such that $x_i + y_i = M$. Now, all recurrences and algorithms introduced above work for this data without any further changes.

2.5.6 Real-valued peak masses

Finally, let us look at the problem of imprecise mass measurements. For the moment, we do not want to look into the details of mass accuracy; this will be covered in Sec. 4.2. Instead, we simply assume that there is some accuracy $\varepsilon > 0$ such that for a measured peak at mass m, the true mass of the ion is somewhere in the interval $[m - \varepsilon, m + \varepsilon]$.

A simple way to deal with this problem, is as follows: When we mirror the spectrum $\mathcal{M}$ as described in Sec. 2.5.3, we identify peaks with masses that differ by at most ε. When building our sequencing graph, we allow that the mass difference between two peaks is in the interval $[\mu(z) - \varepsilon, \mu(z) + \varepsilon]$ for some character $z \in \Sigma$.

The above is the conceptually simplest solution to the problem, but unfortunately, it has some shortcomings: We cannot deal with a series of three or more peaks in the measured spectrum where any two consecutive peak masses have mass difference below ε. Such peak series may arise from mirroring the spectrum. Also, mass error can accumulate, see Exercise 2.17. All of the above "problems" are somewhat academic, and not to be expected often for real-world data. You might be able to come up with a solution for the first problem after reading Chapter 4. We will come back to the second issue in Chapter 8.

2.6 Posttranslational 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*. One example is the oxidization of methionine, that happens spontaneous during the analysis; so, experimentalists often make sure that *all* methionine in the sample is oxidized. Another example is carboxamidomethyl cysteine (CamC), where cysteine reacts with iodoacetamide. See Table 2.2. Note that fixed modifications make

symb.	modified amino acid	molecular formula	mass (Da)
C	carboxamidomethyl cysteine	$C_5H_8N_2O_2S_1$	160.0307*xx*
M	methionine sulfoxide	$C_5H_9N_1O_2S_1$	147.0354*xx*

Table 2.2: Important fixed modifications for amino acids. **[TODO: ANYTHING ELSE?]**

symbol	modified amino acid	molecular formula	mass (Da)
pS	phosphorylated serine	$C_3H_5N_1O_6P$	166.9984*xx*
pT	phosphorylated threonine	$C_4H_7N_1O_6P$	181.0140*xx*
pY	phosphorylated tyrosine	$C_9H_9N_1O_6P$	243.0297*xx*

Table 2.3: Post-Translational Modifications: Phosphorylation of serine, threonine, and tyrosine. **[TODO: CALCULATE MASSES]**

peptide *de novo* sequencing neither simpler nor more complicated, and this is also true for database searching presented in the next chapter: We simply replace one molecular formula of the character by a different one.[3] We do not introduce a new symbol for the modified amino acids, as they are an artifact of the experimental setup, but have no biological meaning.

The second type of modifications are *variable* modifications, enlarging the alphabet of amino acids that we have to look at: *Posttranslational 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 $+PO_4$ and a mass modification of $+79.96633x$ for the affected amino acid. Note that 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, see Table 2.3. Other common posttranslational modifications are pyroglutamic acid replacing glutamine (Q) with mass change $-17.0306xx$ Da; deamidation of glutamine (Q) or asparagine (N) with mass change $+0.9847xx$ Da; and carboxylation of aspartic acid (D) or glutamic acid (E) with mass change $+44.0098xx$ Da. We may also include the methylated form of some amino acids, such as methylated arginine (R^*) with molecular formula $C_6H_{12}N_4O_1$, and doubly methylated arginine (R^{**}) with molecular formula $C_6H_{12}N_4O_1$. **[TODO: INCLUDE WHICH? STREAMLINE!]**

See DeltaMass[4] compiled by Ken Mitchelhill, for a comprehensive list of modifications.

Another common posttranslational 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 14.

2.7 Historical notes and further reading

Mass spectrometry experts still sequence peptides "by hand" — see Seidler, Zinn, Boehm, and Lehmann [207] for a recent review on peptide *de novo* sequencing. At present, automated methods for peptide sequencing are no match for the human experts. Taking into account that the vast majority of peptides that are analyzed each day, are in fact sequenced by automated

[3] 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 application, this subtlety appears to be irrelevant.

[4] `http://www.abrf.org/index.cfm/dm.home`

methods and never looked at by an expert, this only shows the need to further improve our methods for peptide *de novo* sequencing.

Our presentation of this chapter loosely follows the paper of Chen, Kao, Tepel, Rush, and Church [39], with some 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 3.12 and 3.13 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 vertex-weighted graphs. To this end, both our presentation (except for Sec. 2.5.4) as well as the literature [9, 39, 48] use the trick of transforming vertex-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 peaks series, as well as the existence or non-existence of such consecutive peaks. We come back to this issue repeatedly throughout Chapters 4 and 8.

The spectrum graph was introduced by Bartels [9] in 1990. Valid paths in spectrum graphs are a particular case of antisymmetric paths. When Dančík *et al*. [48] cast the peptide *de novo* problem onto the ANTISYMMETRIC LONGEST PATH problem they noted that, in general, this is an NP-complete problem [87]. 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 [39]. In 2011, Andreotti *et al*. [3] presented a faster method for finding longest antisymmetric paths in spectrum graphs in practice, based on Lagrangian relaxation.

There is a huge number of approaches that were proposed throughout the years for *de novo* sequencing of peptides, both before the year 2000 [9, 48, 75, 76, 106, 111, 202, 226, 242] as well as after that year [1, 17, 18, 80, 82, 119, 120, 156, 225]. Early approaches [202] 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 [106, 242] but did not prove very successful, in particular because a correct sequence prefix could be pruned due to peaks missing in the measured spectrum. We will come back to the problem of peptide *de novo* sequencing in Chapter 8: See Sec. 8.2 for a description of the PEAKS algorithm, and Sec. 8.7 for some more detail on other peptide *de novo* sequencing approaches.

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 14: It turns out that this is a computationally hard problem even for the peak counting score.

The antibiotic Actinomycin D [218] discovered in 1940 by Waksman and Woodruff [231]. It is isolated from *Streptomyces* soil bacteria. Another well-known nonribosomal peptide is glutathione, part of the antioxidant defenses of aerobic organisms. Nonribosomal peptide are often cyclic, although linear such peptides are also common. Sequencing a complete nonribosomal peptide using MS can be much more complicated than simply reading off its sequence from a tandem mass spectrum, see Sec. 16.2.

2.8 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 parent

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. Give reason 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 $\mathscr{M}(s) = \mathscr{M}(s')$.

2.4 For Σ from Exercise 2.2, find a string that generates the fragmentation spectrum $\mathscr{M} = \{0,2,3,5,9,11,12,14\}$, where the parent 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 $\mathscr{M}(s) = \mathscr{M}$ for a given set of masses $\mathscr{M}$. 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 $\mathscr{M} = \{m_1,\ldots,m_n\}$. 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 $m \in \mathscr{M}$ 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. Proof 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* Proof 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 $\mathscr{M} := \{0,2,7,8,9,14,16,17,22,24\}$ with parent 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 $\mathscr{M}$ with $0, M \in \mathscr{M}$, such that $m \in \mathscr{M}$ implies $M - m \in \mathscr{M}$ for parent mass M. In addition, we are given a prefix score $w_1 : \mathscr{M} \to \mathbb{R}$ and a suffix score $w_2 : \mathscr{M} \to \mathbb{R}$: 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

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

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

2.13 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.

2.14 Show that

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

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

2.15 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 parent 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.

2.16* Let $G = (V,E)$ be a spectrum graph for some set of masses $\mathcal{M} = \{x_0,\ldots,x_n,y_n,\ldots,y_0\}$ with $x_i + y_i = M$ for all $i = 0,\ldots,n$. Let $w : V \to \mathbb{R}$ be arbitrary *vertex weights*. We define the *valid length* of a path to be the sum over all vertex 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 analog to (2.8) that can be used to compute the maximum valid length of any path in G.

2.17 Find a series of b ion peak masses where, for mass error $\varepsilon = 0.5\,\text{Da}$, 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.

2.18*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 outwards.