Inferring peptide composition from molecular formulas

Sebastian Böcker[1,2] and Anton Pervukhin[1]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany,
sebastian.boecker@uni-jena.de, apervukh@minet.uni-jena.de
[2] Jena Centre for Bioinformatics, Jena, Germany

Abstract. With the advent of novel mass spectrometry techniques such as Orbitrap MS, it is possible to determine the exact molecular formula of an unknown molecule solely from its isotope pattern. But for protein mass spectrometry, one is facing the problem that many peptides have exactly the same molecular formula even when ignoring the order of amino acids. In this work, we present an efficient method to determine the amino acid composition of an unknown peptide solely from its molecular formula. Our solution is based on efficiently enumerating all solutions of the multi-dimensional equality constrained integer knapsack problem.

1 Introduction

Novel mass spectrometry techniques allow us to determine the mass of a sample molecule with very high accuracy of 5 ppm (parts-per-million), and sometimes below 1 ppm [9, 10]. These techniques are increasingly coupled with high throughput separation techniques such as (Ultra) High Performance Liquid Chromatography, and have become one preferred method for the analysis of peptides [8] and metabolites [14]. In proteomics, this is of particular interest to detect post-translational modifications [13], or non-ribosomal peptides that are not directly encoded in the genome [2]. With the advent of new MS instruments such as Orbitraps, mass spectra with very high mass accuracy will be routinely acquired for protein identification and quantification in the near future.

It has been known for almost two decades that one can infer the molecular formula of a sample molecule solely from its isotope pattern [7,12]. But only recently, measurement accuracy has increased to a point where this analysis is feasible for sample molecules with mass of 1000 Dalton and above [5]. In addition, efficient methods had to be developed to carry out the computational analysis for larger molecules in reasonable running time [5, 6]. The upper mass limit of molecules that allow for this interpretation, is ever increasing due to improvements in existing MS techniques as well as the development of new ones.

Given an isotope pattern of an unknown peptide, one can decompose its monoisotopic mass [6] and then score amino acid decompositions with regards to their theoretical isotope pattern [5]. Clearly, this is a non-trivial problem since there exist about $3.96 \cdot 10^{11}$ amino acid decompositions with mass up to 2500 Dalton. Unfortunately, many peptides have exactly the same molecular formula even when ignoring the order of amino acids: Besides leucine and isoleucine, the smallest non-trivial example are peptides consisting of two glycine vs. a single asparagine, both with molecular formula $C_4H_8N_2O_3$. Using the above technique, we repeatedly score amino acid decompositions with identical molecular formula and, hence, identical isotope pattern. On the other hand, the sample may be contaminated by metabolite molecules that have a molecular formula which cannot be explained by any peptide. By determining the molecular formula for the isotope pattern, we can effectively sort out such contaminants. So, it is much more efficient to first determine the molecular formula of a sample from its isotope pattern, and then to compute all amino acid compositions that match the molecular formula.

Our contributions. Our input is the molecular formula of an unknown peptide. Our goal then is to find all amino acid compositions that match the given molecular formula. We formulate the problem as a joint decomposition of a set of queries or, equivalently, as a multi-dimensional equality constrained integer knapsack problem [1]. Our queries are the number of carbon, hydrogen, and other atoms that make up the molecule. We present the dimension reduction method that reduces a multi-dimensional problem to a one-dimensional decomposition problem, which in turn can be efficiently solved using methods presented in [6]. We also provide an experimental evaluation of the algorithm's running time, both on simulated data and peptides from experimental mass spectra. We find that our *mixed matrix* approach is the fastest method for enumerating solutions, and is one to two orders of magnitude faster than the runner-up algorithm.

2 Preliminaries

Proteins and peptides are made up from the six elements hydrogen (symbol H), carbon (C), nitrogen (N), oxygen (O), and sulfur (S). The unusual amino acid selenocysteine also contains the element selenium (Se). For each amino acid we know its exact molecular formula, such as $C_3H_7N_1O_2$ for alanine. When an amino acid is added to a peptide chain, it looses a water molecule. In the following, we concentrate on amino acid *residues* that are missing a water molecule H_2O. See Table 1 for the molecular formulas of amino acid residues. Note that leucine and isoleucine have identical molecular formula and cannot be told apart by mass spectrometry. In the following, we treat these two amino acids as one, and talk about 19 standard amino acids.

symb.	TLC	a.a. residue	molecular formula	symb.	TLC	a.a. residue	molecular formula
A	Ala	alanine	$C_3H_5N_1O_1$	M	Met	methionine	$C_5H_9N_1O_1S_1$
C	Cys	cysteine	$C_3H_5N_1O_1S_1$	N	Asn	asparagine	$C_4H_6N_2O_2$
D	Asp	aspartic acid	$C_4H_5N_1O_3$	P	Pro	proline	$C_5H_7N_1O_1$
E	Glu	glutamic acid	$C_5H_7N_1O_3$	Q	Gln	glutamine	$C_5H_8N_2O_2$
F	Phe	phenylalanine	$C_9H_9N_1O_1$	R	Arg	arginine	$C_6H_{12}N_4O_1$
G	Gly	glycine	$C_2H_3N_1O_1$	S	Ser	serine	$C_3H_5N_1O_2$
H	His	histidine	$C_6H_7N_3O_1$	T	Thr	threonine	$C_4H_7N_1O_2$
I	Ile	isoleucine	$C_6H_{11}N_1O_1$	V	Val	valine	$C_5H_9N_1O_1$
K	Lys	lysine	$C_6H_{12}N_2O_1$	W	Trp	tryptophan	$C_{11}H_{10}N_2O_1$
L	Leu	leucine	$C_6H_{11}N_1O_1$	Y	Tyr	tyrosine	$C_9H_9N_1O_2$
U	Sec	selenocysteine	$C_3H_5N_1O_1Se_1$				

Table 1. Amino acid residues with 3-letter-code (TLC), symbol, and molecular formula.

In this paper, we want to find all amino acid compositions that match a given molecular formula. To approach this problem, we can use branch-and-bound search by adding amino acids as long as for each element, the resulting molecule contains at most as many atoms as the input molecule, and output exact hits. Alternatively, we can compute the molecular formulas of all amino acid compositions up to a certain mass during preprocessing, and use hashing to efficiently search this list. Particularly the latter approach suffers from the large number of amino acid decompositions, see above.

We want to approach the problem of decomposing a molecular formula over the amino acid alphabet, as a multi-dimensional equality constrained integer knapsack problem. Recall that we ignore isoleucine in our presentation. Now, we can formulate our problem as a matrix multiplication $Ax = b$ where A is a matrix containing multiplicities of all elements in amino acids, x is the 19-dimensional vector we search for, and b is the input molecular formula over the elements CHNOS. But one can use one more trick to further simplify the problem: only amino acids methionine and cysteine contain sulfur. So, if our input molecular formula contains k sulfur atoms, we first try to distribute these between methionine and cysteine, iterating over all possibilities $\mathrm{M}_0\mathrm{C}_k$, $\mathrm{M}_1\mathrm{C}_{k-1}, \dots,$ $\mathrm{M}_k\mathrm{C}_0$. In each case, we reduce the input molecular formula accordingly, and we try to decompose the resulting formulas over the remaining 17 amino acids A, D, E, F, G, H, K, L, N, P, Q, R, S, T, V, W, Y:

$$A := \begin{pmatrix} 3&4&5&9&2&6&6&6&4&5&5&6&3&4&5&11&9 \\ 5&5&7&9&3&7&12&11&6&7&8&12&5&7&9&10&9 \\ 1&1&1&1&1&3&2&1&2&1&2&4&1&1&1&2&1 \\ 1&3&3&1&1&1&1&1&2&1&2&1&2&2&1&1&2 \end{pmatrix}, \quad b = \begin{pmatrix} \#\ \mathrm{C} \\ \#\ \mathrm{H} \\ \#\ \mathrm{N} \\ \#\ \mathrm{O} \end{pmatrix}, \quad Ax = b \tag{1}$$

Here, x is a 17-dimensional vector representing the remaining amino acid residues.

3 Single-dimensional Integer Knapsack

We first consider the single-dimensional equality constrained integer knapsack [1]

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b \tag{2}$$

where a_j are integer-valued coefficients usually satisfying $a_j \geq 0$, and $b \geq 0$. We search for all solution vectors $x = (x_1, \dots, x_n)$ such that all x_j are non-negative integers. We start with an important observation: if there exist indices i, j with $a_i > 0$ and $a_j < 0$, and if (2) has at least one solution, then there is an *infinite* number of solutions. In the following, we assume $a_j \geq 0$ for all j.

One can use dynamic programming to efficiently compute all solutions of (2) [6]: We choose a maximal integer B that we want to decompose, and construct a bit table of size $n \times B$ during preprocessing. Using this table, we can efficiently find all solutions (2) for all queries $b \leq B$. An alternative method for finding all solutions uses a Extended Residue Table of size $n \cdot a_1$, see [6] for details. Here, every decomposition is constructed in time $O(na_1)$ independent of the input b. In addition, we do not have to choose a maximal integer B during preprocessing. This latter method also appears to be faster in practice. Finally, we can count the exact number of decompositions $\gamma(b)$ of the integer b using a dynamic recurrence similar to the bit table mentioned above, see again [6].

The number of decompositions over coprime integers $a_1, \dots, a_n$ asymptotically behaves like a polynomial of degree $n-1$ in b [15]:

$$\gamma(b) \sim \tfrac{1}{(n-1)!\, a_1 \cdots a_n}\, b^{n-1}. \tag{3}$$

We can use this formula or the more precise version from [3], to approximate the number of amino acid decompositions $\hat{\gamma}(M, \epsilon)$ with *real mass* in the interval $[M, M+\epsilon]$, over the 19 standard amino acids:

$$\hat{\gamma}(M, \epsilon) \approx\ 1.12687 \cdot 10^{-55}\, \epsilon\, M^{18} + 2.29513 \cdot 10^{-51}\, \epsilon\, M^{17} + 2.16611 \cdot 10^{-47}\, \epsilon\, M^{16} \tag{4}$$

Unfortunately, approximation (4) is very inaccurate for masses below 10 000 Da. To this end, we use an improved version of (4) with eight leading coefficients, see the Appendix. Note also that the true number of molecules is oscillating with high intensity. In Fig. 1, we plot the number of decompositions with mass up to M, and the true and approximate number of decompositions with mass M for bin width $\epsilon = 0.001\,\text{Da}$, for masses $0 \le M \le 2500$ Da.

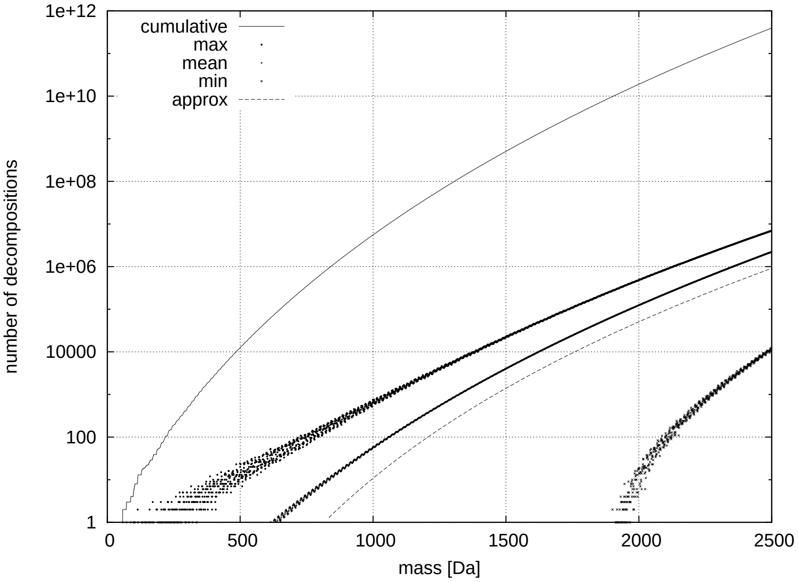

Fig. 1. Number of amino acid decompositions with mass up to M (*cumulative*), true and approximate (*approx*) number of decompositions with mass M to $M + \epsilon$ for $\epsilon = 0.001\,\text{Da}$. Maxima (*max*), mean and minima (*min*) of the true number calculated using bin width 1 Da.

4 Multi-dimensional Integer Knapsack

We now generalize (2) to the multi-dimensional equality constrained integer knapsack problem: We want to find all solutions of the equation $Ax = b$ for $A = (a_{i,j})_{1 \le i \le d, 1 \le j \le n}$ where $a_{i,j}$ are integer-valued coefficients satisfying $a_{i,j} \ge 0$, and $b_i \ge 0$. We search for all solution vectors $x = (x_1, \dots, x_n)$ such that all x_j are non-negative integers. This corresponds to d one-dimensional knapsack equations (2) that we want to solve simultaneously, and it is a special case of a Diophantine equation, where all entries are non-negative.

A simple algorithm to compute all solutions of $Ax = b$, is to choose one row $i \le d$ as the master row, then to find all solutions of the one-dimensional integer knapsack $a_{i,1}x_1 + \cdots + a_{i,n}x_n = b_i$ and, finally, to test for each solution of the master equation if the solution also satisfies the other rows of matrix A. We call this the *naïve decomposition* algorithm. However, this involves generating many decompositions unnecessarily. Decompositions of the master equation can be found by recursing through the Extended Residue table, see [4]. Böcker *et al.* [4] also present a method that can be seen as an intermediate between the naïve decomposition algorithm, and the method presented below: The *multiple decomposition* algorithm also chooses a master equation to decompose, but tests during recursion if all other equations of $Ax = b$ besides the master equation can still be satisfied using the current partial solution. If this is no longer possible, then it stops and discards the current partial solution.

Equation (3) tells us that the number of solutions increases with a polynomial of degree $n-1$. So, it seems advisable to lower n as much as possible, if we can do so. In fact, the multi-dimensional knapsack gives us an opportunity to lower n: To this end, we apply a Gaussian elimination to matrix A to find a lower triangular matrix $L \in \mathbb{R}^{d\times d}$ and an upper triangular matrix $R = (r_{i,j}) \in \mathbb{R}^{d\times n}$ such that $A = LR$. Then, $Ax = b$ if and only if $Rx = L^{-1}b =: b'$ where L^{-1} is known. Every solution of $Ax = b$ must hence satisfy the *bottom equation* of R,

$$0 \cdot x_1 + \cdots + 0 \cdot x_{d-1} + r_{d,d}x_d + \cdots + r_{d,n}x_n = b'_d \tag{5}$$

that has at most $n-d+1$ non-zero coefficients. We now search for all solutions of the bottom equation, and we test for each one if it is also a solution of $Ax = b$.

If all entries in A are integers, we can easily guarantee the same to be true for the output matrix R. But it should be understood that even if $a_{i,j} \geq 0$ holds for all coefficients, we cannot guarantee $r_{i,j} \geq 0$ for all coefficients after Gaussian elimination. In particular, there may be negative coefficients in the bottom equation. But as we have learned in Sec. 3, this implies that there is an infinite number of solutions for the bottom equation, provided that there exists at least one solution. Hence, we have to avoid that negative coefficients appear in the bottom equation. Can we find Gaussian eliminations schemes where all coefficients of the bottom equation are non-negative? To do so, we may have to permute the columns and rows of A: We choose a permutation π of the rows of A, and a permutation σ of the columns of A that brings d columns to the front but ignores the remaining $n-d$ columns. We have $d!$ possibilities to choose π, and $(n-d+1)\cdots n$ possibilities to choose d front rows of A in σ.

We use the following simple version of the Gaussian elimination algorithm in our computations: Assume that rows and columns have been swapped in matrix A. Set $\tilde{L} := L^{-1}$, we will compute $\tilde{L}$ instead of L. We initialize $\tilde{L} = (l_{i,j}) \leftarrow I$ as the identity matrix and $R \leftarrow A$. We iterate the following for $i = 1,\ldots,d-1$. For rows $i' = i+1,\ldots,d$ and columns $j = i,\ldots,n$ we define the new submatrix $r'_{i',j} = r_{i',i}r_{i,j} - r_{i,i}r_{i',j}$. Then, $r'_{i',i} = 0$ must hold. Similarly, for rows $i' = i+1,\ldots,d$ and columns $j' = 1,\ldots,n$ we compute the new submatrix $l'_{i',j'} = r_{i',i}l_{i,j'} - r_{i,i}l_{i',j'}$. But if $r_{i,i} = 0$ this operation reduces the rank of our matrix R, and the resulting matrix is no longer equivalent to our input matrix. In consequence, we Stop if we encounter the case $r_{i,i} = 0$.

In case we do not drop out off the elimination algorithm, we test if all entries of the bottom equation are non-positive: In this case, we negate the bottom equation. Finally, we check if $r_{d,j} \geq 0$ holds for all $j = d,\ldots,n$: Otherwise, we have to discard R, L^{-1}. Different permutations π might lead to the same bottom equation if σ is kept constant, so we finally have to sort out duplicate bottom equations. We end up with a list of elimination matrix pairs R, L^{-1} that all allow us to compute decompositions for the multi-dimensional problem using their bottom equations: Assume that for one such pair R, L^{-1} we are given a input vector b. First, we apply the row permutation π to b, that was used to generate A. Then, we find all solutions of the equation $Rx = L^{-1}b$ as follows: We compute $b' \leftarrow L^{-1}b$, and we use one of the decomposition techniques for single-dimensional integer knapsacks on the bottom equation of $Rx = b'$. For every decomposition $(x_d,\ldots,x_n)$, we iterate $i = d-1, d-2,\ldots,1$ and compute entry x_i using row i of $Rx = b'$. We test if $x_i \geq 0$ and if x_i is integer; otherwise, we discard the decomposition. Finally, we apply the inverse column permutation σ^{-1} to x. Doing so for all decompositions of the bottom equation, guarantees that we find all solutions of $Ax = b$. On the other hand, many decompositions can be generated "in vain" because these are no solutions of $Ax = b$.

5 Mixed Matrix Approach

We have used Gaussian elimination for all row and column permutations, a total of $1 \cdot 2 \cdot 3 \cdot 4 \cdot 14 \cdot 15 \cdot 16 \cdot 17 = 1370880$ possibilities. In 43176 cases, the reduction scheme generated a bottom equation with non-negative entries. After discarding identical bottom equations, we ended up with only 19 matrix pairs R, L^{-1}, see Table 2 in the Appendix.

Now, one question remains: Which of these matrix pairs R, L^{-1} is "the best"? Different matrix pairs usually differ in the number of decompositions that are generated in vain, and that have to be discarded. If we use efficient decomposition techniques from [6] for single-dimensional decomposition, then we can guarantee that running time for generating decompositions is actually linear in the number of decompositions. Then, the number of discarded decompositions is an excellent indicator for the quality of a matrix pair.

In our evaluations we use the notion of *competitive ratio*: the ratio between the number of true decompositions, over the number of decompositions generated by a particular matrix pair. Using a training set of molecular formulas to decompose, we can filter out matrices that generate too many additional candidates. To find the exact number of decompositions of a particular matrix, we can use the dynamic programming techniques mentioned in Sec. 3.

Although being an ideal indicator for evaluation purposes, in application one would like to avoid the explicit calculation of the number of decompositions, because this can be very time consuming. Therefore, the following question inevitably arises: Can we estimate the number of discarded decompositions, without actually calculating decompositions? To this end, set $b'_d := \sum_{k=1}^{d} l_{d,k} b_k$ as the number we actually want to decompose in the bottom equation (5). Recall that the number of decompositions of b over n coprime integers asymptotically behaves like a polynomial of degree $n-1$, see (3). So, we define

$$\tilde{l}(b) := \frac{1}{(m-1)!\, r_{n-m+1,d} \cdots r_{n,d}} \left(\sum\nolimits_{k=1}^{d} l_{d,k} b_k \right)^{m-1} \tag{6}$$

as our *indicator*, where m is a number of non-zero values in the bottom row of R.

Our *mixed matrix* approach now works as follows: Given a vector b to decompose, we compute the indicators $\tilde{l}(b)$ for every matrix pair L^{-1}, R and choose the matrix pair with the smallest indicator. Then, we use the bottom equation of the corresponding matrix R to actually decompose b, see the previous section.

6 Experimental Results

To evaluate our approach we proceed as follows: First, we calculate the competitive ratios of all matrices and filter out those that generate too many candidates in vain. The number of decompositions is not the only factor affecting the running time. In addition, the time required to filter out incorrect solutions may vary for different matrix pairs. To better estimate the actual running time for a particular matrix pair, we will apply a slight correction to our indicator. We also compare running times of the mixed matrix approach to several other algorithms.

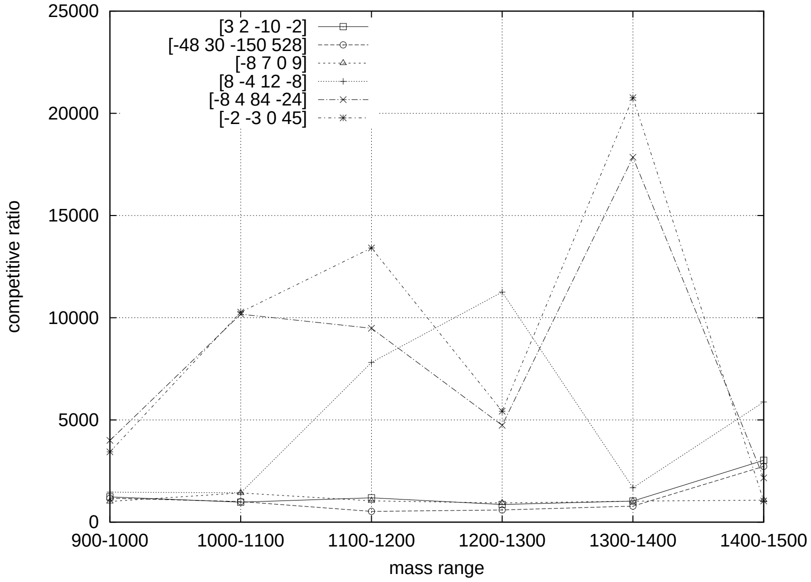

Fig. 2. Competitive ratios of six matrices with the lowest competitive ratios. Average competitive ratio for mass bins of width 100 Da.

Datasets. For our evaluations we use two datasets. The first dataset with 6000 peptides has been simulated by in-silico digestion (trypsin) using the Swiss-Prot database (release 56.5), eliminating duplicates. We select peptides with masses between 900 and 1500 Da, and for each mass range of 100 Da we randomly choose 1000 peptides. The second dataset consists of 99 peptides from *de novo* interpretations of experimental mass spectra, acquired on quadrupole ion-trap mass spectrometer. These peptides range in mass from about 900 to 2000 Da.

Choosing Good Matrix Pairs. For each matrix pair we calculate the competitive ratio for all peptides in the simulated dataset, so that we can filter out matrix pairs that generate too many false positives. We calculate the average competitive ratio over all peptides for bins of size 100 Da. In Fig. 2 we have depicted the competitive ratios of the six best matrices. Matrices are labeled by the last row of the matrix L^{-1}, see Table 2 in the Appendix. One can see that three matrices show outstanding competitive ratios. For the remaining 13 matrices, we find that the average competitive ratio never drops below 3900 for any matrix and any mass bin of size 100 Da. See Fig. 6 in the Appendix for the competitive ratios of seven more matrices.

We selected the three matrix pairs with the best competitive ratios for further evaluation. These matrices can be found in Sec. 8.3 of the Appendix. For each matrix pair we calculate the indicator $\tilde{l}(b)$ and compare it with the actual running time for the input vector b. In Fig. 3 we have plotted these values for all peptides from our first dataset with mass 1000–1100 Da. We observe an almost linear correlation between indicators and running times. We also observe a slight shift of the intercept of the linear fit over the y-axis for various matrices. This corresponds to the differences in running times required for filtering false decompositions. We derive an affine indicator correction from this experimental data, using a linear fit with the least squares criterion, see Fig. 3. Peptides from other mass ranges show a similar behavior (data not shown).

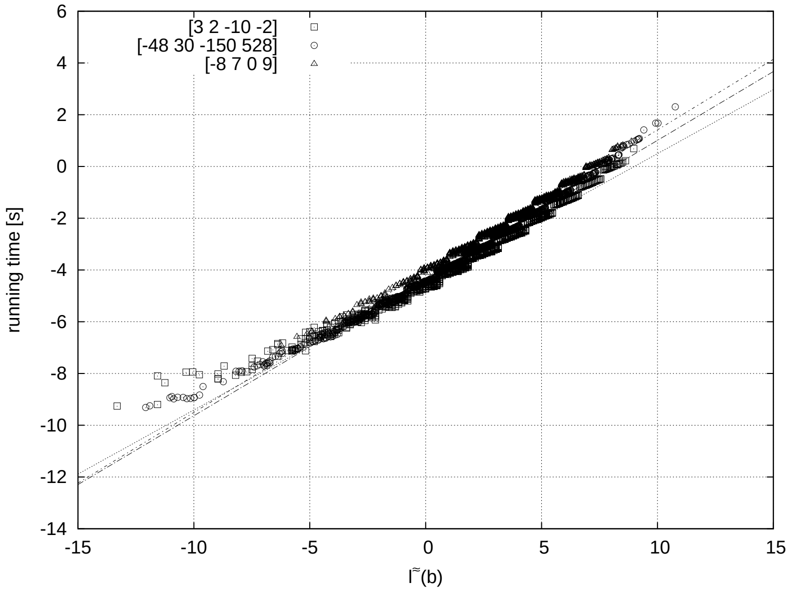

Fig. 3. Indicators of running times calculated for peptides in the mass range 1000–1100 Da. Values are plotted in a log-log scale. Lines show results of the least squares fits.

To find the matrix pair that we actually use in our *mixed matrix* approach to decompose b, we apply the linear correction to the indicator $\tilde{l}(b)$ and choose the matrix with the minimal value. Clearly, this is not necessarily the matrix pair with minimal running times. We want to evaluate how often we choose a suboptimal matrix pair from these three pairs, see Fig. 7 in the Appendix: For the simulated dataset, we choose the correct matrix pair in more than 97.5 % of the cases, resulting in an overall running time increase of less than 0.4 %. Results on the real data are similar: in less than 5 % of the cases a suboptimal matrix is chosen, resulting in a total running time increase of 0.8 %.

Comparison with other methods. Finally, we want to evaluate how good the mixed matrix method works compared to branch-and-bound searching, the naïve decomposition algorithm [6], and the multiple decomposition algorithm [4]. In all cases, we distribute sulfur atoms between methionine and cysteine.

The branch-and-bound search first tries all possibilities for alanine and branches, then does the same for aspartic acid, and so on until we reach the last amino acid tyrosine. The naïve decomposition algorithm simply uses one of the rows of matrix A to compute decompositions, and then test if any such decomposition satisfies $Ax = b$. Both for this and the multiple decomposition algorithm, we use Extended Residue Table to compute decompositions [6]. In fact, there exist four different flavors of the latter two algorithms, as we can choose one of the rows of matrix A from (1) as our master row. For these two methods, we only report the best results of the four possibilities.

Fig. 4 and 5 show running times of these approaches on simulated and real data. Running times for the naïve decomposition algorithm are significantly worse than those of all other approaches and, hence, omitted. For both datasets our mixed matrix approach significantly outperforms the second best approach, branch-and-bound searching. We observe a 16-fold speedup over the branch-and-bound algorithm, and a 35-fold speedup over the multiple decomposition algorithm on

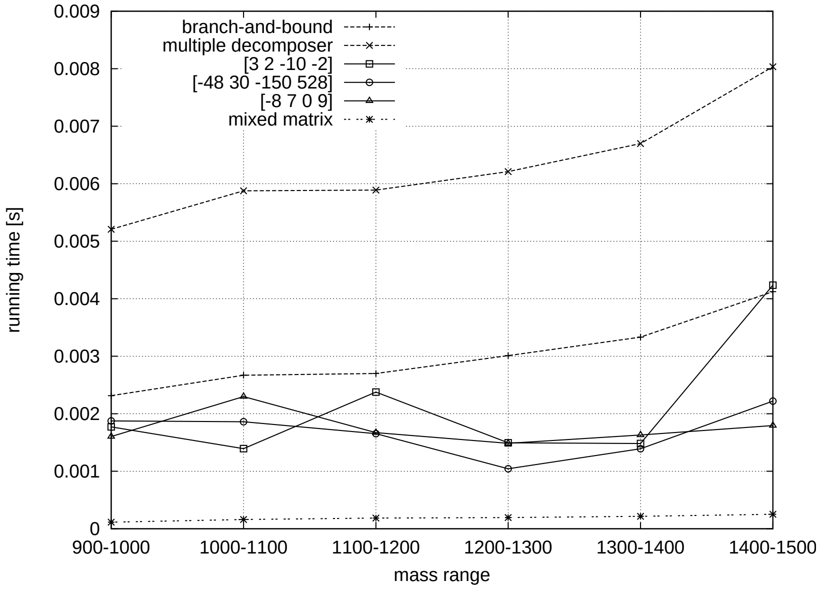

Fig. 4. Running times of the algorithms (in seconds) for simulated data. Running times are calculated per decomposition and averaged over all peptides in the mass range. We also report the performance of our method with only one particular matrix pair applied.

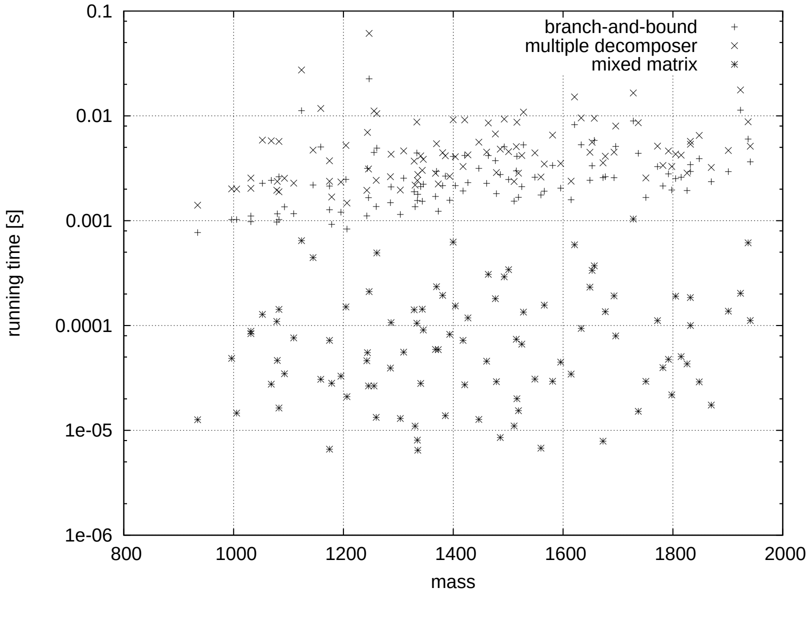

Fig. 5. Running times of the algorithms (in seconds) for real data.

average. Running times of the mixed matrix approach range from 0.11 to 0.25 milliseconds per decomposition, measured for the simulated dataset. On the real dataset, speedup of the mixed matrix approach reaches 67-fold over the branch-and-bound algorithm. We also observe that the mixed matrix approach significantly outperforms each individual matrix pairs, see Fig. 4.

We have also evaluated matrix pairs with five rows, that include sulfur as a decomposable element. Performance for these matrix pairs was in all cases significantly slower than what we have presented above.

All algorithms were implemented in C++, and running times measured on an AMD Opteron-275 2.2 GHz with 6 GB of memory running Solaris 10.

7 Conclusion

We have presented an efficient method to enumerate all solutions of a multi-dimensional equality constrained integer knapsack problem. We have applied our method to the problem of finding all amino acid compositions with a given molecular formula. First results on both simulated and real data show the outstanding performance of our *mixed matrix* approach which is one to two orders of magnitude faster than the runner-up method. We are currently conducting further comparisons of our algorithm with other related methods such as [11], that solve linear Diophantine systems with negative coefficients.

We can easily include more matrix pairs for computing decomposition, what seems advisable for molecular formulas with mass above 1400 Da. We can speed up the decomposition process by eliminating duplicates in the bottom row of R, distributing the resulting number between the “merged” amino acids. Finally, note that the first row of R contains only positive entries, resulting in upper bounds for amino acids that can be dynamically updated during backtracking.

Clearly, our method can be used for any application where we have to enumerate all solutions of a multi-dimensional equality constrained integer knapsack [1]. This is necessary whenever finding an optimal solution of the knapsack cannot be modeled via a simple linear or quadratic objective function. For example, the mixed matrix method can be used to speed up the search for a molecular formula of an unknown sample molecule, as proposed in [4].

Acknowledgments. AP supported by Deutsche Forschungsgemeinschaft (BO 1910/1). We thank Andreas Bertsch from the Division for Simulation of Biological Systems of the Tübingen University for providing the peptide dataset.

References

1. K. Aardal and A. K. Lenstra. Hard equality constrained integer knapsacks. In *Proc. of Integer Programming and Combinatorial Optimization (IPCO 2002)*, volume 2337 of *Lect Notes Comput Sci*, pages 350–366. Springer, Berlin, 2002.
2. N. Bandeira, J. Ng, D. Meluzzi, R. G. Linington, P. Dorrestein, and P. A. Pevzner. De novo sequencing of nonribosomal peptides. In *Proc. of Research in Computational Molecular Biology (RECOMB 2008)*, volume 4955 of *Lect Notes Bioinform*, pages 181–195. Springer, Berlin, 2008.
3. 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.
4. S. Böcker, M. Letzel, Zs. Lipták, and A. Pervukhin. Decomposing metabolomic isotope patterns. In *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2006)*, volume 4175 of *Lect Notes Comput Sci*, pages 12–23. Springer, Berlin, 2006.
5. S. Böcker, M. Letzel, Zs. Lipták, and A. Pervukhin. SIRIUS: Decomposing isotope patterns for metabolite identification. *Bioinformatics*, 25(2):218–224, 2009.
6. S. Böcker and Zs. Lipták. A fast and simple algorithm for the Money Changing Problem. *Algorithmica*, 48(4):413–432, 2007.

7. A. Fürst, J.-T. Clerc, and E. Pretsch. A computer program for the computation of the molecular formula. *Chemom Intell Lab Syst*, 5:329–334, 1989.
8. W. Haas, B. K. Faherty, S. A. Gerber, J. E. Elias, S. A. Beausoleil, C. E. Bakalarski, X. Li, J. Ville, and S. P. Gygi. Optimization and use of peptide mass measurement accuracy in shotgun proteomics. *Mol Cell Proteomics*, 5.7:1326–1337, 2006.
9. F. He, C. L. Hendrickson, and A. G. Marshall. Baseline mass resolution of peptide isobars: A record for molecular mass resolution. *Anal Chem*, 73(3):647–650, 2001.
10. J. V. Olsen, L. M. F. de Godoy, G. Li, B. Macek, P. Mortensen, R. Pesch, A. Makarov, O. Lange, S. Horning, and M. Mann. Parts per million mass accuracy on an orbitrap mass spectrometer via lock mass injection into a c-trap. *Mol Cell Proteomics*, 4:2010–2021, 2006.
11. D. Papp and B. Vizvári. Effective solution of linear diophantine equation systems with an application in chemistry. *J Math Chem*, 39(1):15–31, 2006.
12. A. L. Rockwood and S. L. Van Orden. Ultrahigh-speed calculation of isotope distributions. *Anal Chem*, 68:2027–2030, 1996.
13. S. Tanner, S. H. Payne, S. Dasari, Z. Shen, P. A. Wilmarth, L. L. David, W. F. Loomis, S. P. Briggs, and V. Bafna. Accurate annotation of peptide modifications through unrestrictive database search. *J Proteome Res*, 7:170–181, 2008.
14. E. von Roepenack-Lahaye, T. Degenkolb, M. Zerjeski, M. Franz, U. Roth, L. Wessjohann, J. Schmidt, D. Scheel, and S. Clemens. Profiling of Arabidopsis secondary metabolites by capillary liquid chromatography coupled to electrospray ionization quadrupole time-of-flight mass spectrometry. *Plant Physiol*, 134(2):548–559, 2004.
15. H. Wilf. *generatingfunctionology*. Academic Press, second edition, 1994. Freely available from `http://www.math.upenn.edu/~wilf/DownldGF.html`.

8 Appendix

8.1 Approximating the number of amino acid decompositions

As noted above, approximating the number of amino acid decompositions using (4) is rather inaccurate. Here, we give a better approximation with eight leading coefficients [3]:

$$\begin{aligned}\hat{\gamma}(M,\epsilon) \approx\ & 1.12687 \cdot 10^{-55}\, \epsilon\, M^{18} + 2.29513 \cdot 10^{-51}\, \epsilon\, M^{17} + 2.16611 \cdot 10^{-47}\, \epsilon\, M^{16} \\ & + 1.25733 \cdot 10^{-43}\, \epsilon\, M^{15} + 5.02375 \cdot 10^{-40}\, \epsilon\, M^{14} + 1.46529 \cdot 10^{-36}\, \epsilon\, M^{13} \\ & + 3.22832 \cdot 10^{-33}\, \epsilon\, M^{12} + 5.48395 \cdot 10^{-30}\, \epsilon\, M^{11}\end{aligned} \tag{7}$$

8.2 Table of conversion matrix pairs

Last row (modified) of the matrix R														lcm	Last row of L^{-1}			
	1	1	1	1	2	2	2	2	4	5	5	6	8	8	8	-4	12	-8
3	4	4	6	7	7	7	7	12	14	14	18	26	39	8	-8	4	84	-24
	1	2	2	2	3	3	4	5	5	6	6	7	8	4	0	4	-8	-4
	1	1	1	1	1	2	3	3	3	4	6	7	8	4	4	0	-4	-4
5	10	10	16	16	17	22	27	29	33	56	58	73	77	8	-80	108	-76	-48
3	5	6	8	10	10	11	12	14	16	23	29	36	39	40	-200	220	60	-120
1	1	1	3	4	4	5	7	7	8	9	18	18	19	10	20	-15	65	-20
1	1	1	1	2	2	3	4	6	7	7	8	15	19	10	0	-5	65	-10
2	4	7	9	12	13	16	19	27	28	31	32	45	49	90	-90	-135	315	1710
	5	6	7	7	12	13	17	20	21	28	31	31	33	1	3	2	-10	-2
	1	2	2	2	3	5	5	6	7	8	16	18	21	12	36	-6	-30	-24
18	25	29	31	32	34	35	62	62	63	76	77	116	117	12	-48	30	-150	528
1	1	3	4	5	6	7	8	8	9	24	27	29	43	3	21	-12	12	-6
2	5	6	11	15	18	24	33	42	43	48	49	73	79	12	-12	-30	30	372
1	5	6	8	9	10	11	13	14	26	45	78	84	113	1	19	-9	0	-6
1	1	1	1	1	2	2	2	2	4	8	12	13	18	6	18	-9	3	-6
5	9	17	25	27	41	53	81	94	101	106	113	171	183	1	-2	-5	0	72
4	7	7	8	11	12	12	16	19	21	28	32	51	66	2	-2	-3	0	45
6	7	9	9	10	10	13	14	14	16	17	19	27	36	2	-8	7	0	9

Table 2. 19 conversion matrix pairs R, L^{-1} with the number of rows $d = 4$. Here, the last row of the matrix R is shown modified: Values have been divided on the least common multiplier (lcm) and sorted in the increasing order; zero entries have been removed.

8.3 Best three matrix pairs

In the first row of each matrix, we give the permutation of amino acids (for R matrices) or elements (for L^{-1} matrices) that have to be applied before or after using the matrices for decomposition. Last rows of the matrices R_2 and R_3 are divided by the least common multipliers 12 and 2 correspondingly.

$$R_1 = \left(\begin{array}{ccccccccccccccccc}
\mathrm{G} & \mathrm{H} & \mathrm{R} & \mathrm{F} & \mathrm{A} & \mathrm{D} & \mathrm{K} & \mathrm{L} & \mathrm{N} & \mathrm{P} & \mathrm{Q} & \mathrm{E} & \mathrm{S} & \mathrm{T} & \mathrm{V} & \mathrm{W} & \mathrm{Y} \\
2 & 6 & 6 & 9 & 3 & 4 & 6 & 6 & 4 & 5 & 5 & 5 & 3 & 4 & 5 & 11 & 9 \\
0 & 4 & -6 & 9 & -1 & 2 & -4 & -6 & 0 & 1 & -1 & 1 & -1 & -2 & -3 & 13 & 9 \\
0 & 0 & -2 & 7 & 1 & 2 & 4 & 2 & 0 & 3 & 1 & 3 & 1 & 2 & 3 & 7 & 7 \\
0 & 0 & 0 & 33 & 7 & 6 & 28 & 20 & 0 & 17 & 7 & 13 & 5 & 12 & 21 & 31 & 31
\end{array}\right) \quad L_1^{-1} = \left(\begin{array}{cccc}
\mathrm{C} & \mathrm{H} & \mathrm{N} & \mathrm{O} \\
1 & 0 & 0 & 0 \\
3 & -2 & 0 & 0 \\
1 & 0 & -2 & 0 \\
3 & 2 & -10 & -2
\end{array}\right)$$

$$R_2 = \left(\begin{array}{ccccccccccccccccc}
\mathrm{H} & \mathrm{R} & \mathrm{W} & \mathrm{F} & \mathrm{G} & \mathrm{A} & \mathrm{K} & \mathrm{L} & \mathrm{N} & \mathrm{P} & \mathrm{Q} & \mathrm{D} & \mathrm{S} & \mathrm{T} & \mathrm{V} & \mathrm{E} & \mathrm{Y} \\
6 & 6 & 11 & 9 & 2 & 3 & 6 & 6 & 4 & 5 & 5 & 4 & 3 & 4 & 5 & 5 & 9 \\
0 & -30 & 17 & 9 & -4 & -9 & -24 & -30 & -8 & -7 & -13 & -2 & -9 & -14 & -19 & -7 & 9 \\
0 & 0 & -88 & -96 & -4 & -24 & -84 & -60 & -8 & -52 & -28 & -32 & -24 & -44 & -64 & -52 & -96 \\
0 & 0 & 0 & 18 & 31 & 32 & 35 & 25 & 62 & 29 & 63 & 116 & 76 & 77 & 34 & 117 & 62
\end{array}\right)$$

$$L_2^{-1} = \left(\begin{array}{cccc}
\mathrm{C} & \mathrm{H} & \mathrm{N} & \mathrm{O} \\
1 & 0 & 0 & 0 \\
7 & -6 & 0 & 0 \\
-8 & -6 & 30 & 0 \\
-48 & 30 & -150 & 528
\end{array}\right)$$

$$R_3 = \left(\begin{array}{ccccccccccccccccc}
\mathrm{F} & \mathrm{W} & \mathrm{Y} & \mathrm{A} & \mathrm{G} & \mathrm{H} & \mathrm{K} & \mathrm{L} & \mathrm{N} & \mathrm{P} & \mathrm{Q} & \mathrm{R} & \mathrm{S} & \mathrm{T} & \mathrm{V} & \mathrm{D} & \mathrm{E} \\
9 & 11 & 9 & 3 & 2 & 6 & 6 & 6 & 4 & 5 & 5 & 6 & 3 & 4 & 5 & 4 & 5 \\
0 & 1 & 0 & -2 & -1 & -1 & -5 & -6 & -2 & -2 & -3 & -6 & -2 & -3 & -4 & -1 & -2 \\
0 & 0 & 9 & 2 & 5 & 1 & -7 & -9 & 10 & 0 & 7 & -9 & 11 & 8 & -4 & 21 & 18 \\
0 & 0 & 0 & 10 & 7 & 14 & 19 & 27 & 14 & 9 & 17 & 36 & 10 & 13 & 16 & 6 & 9
\end{array}\right) \quad L_3^{-1} = \left(\begin{array}{cccc}
\mathrm{C} & \mathrm{H} & \mathrm{O} & \mathrm{N} \\
1 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 \\
1 & -2 & 9 & 0 \\
-8 & 7 & 0 & 9
\end{array}\right)$$

8.4 Additional figures

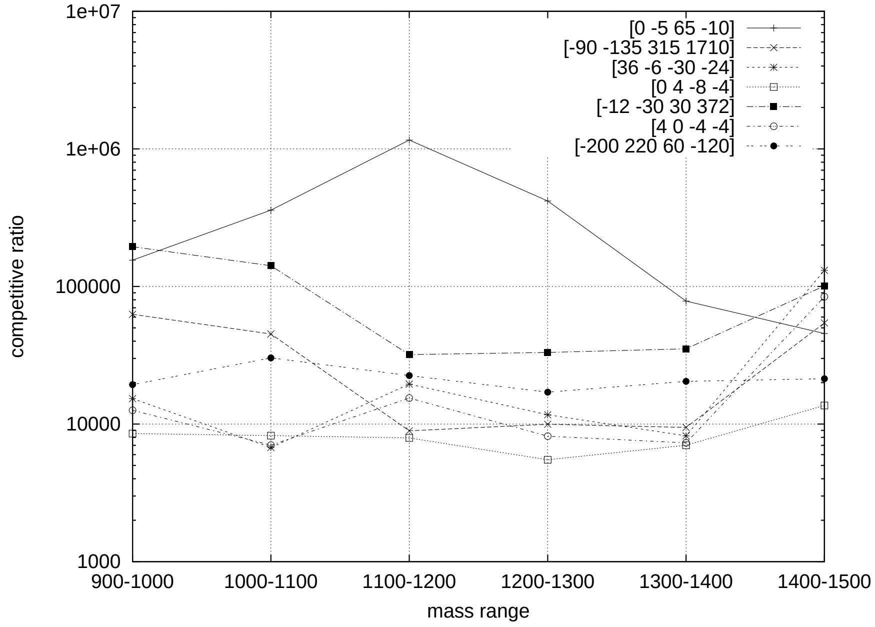

Fig. 6. Competitive ratios of seven matrices with rather high competitive ratios compared to the matrices depicted in Fig. 2. Values are calculated on average for mass bins of width 100 Da. Competitive ratios for the remaining six matrices have comparable or higher values than the matrices shown here.

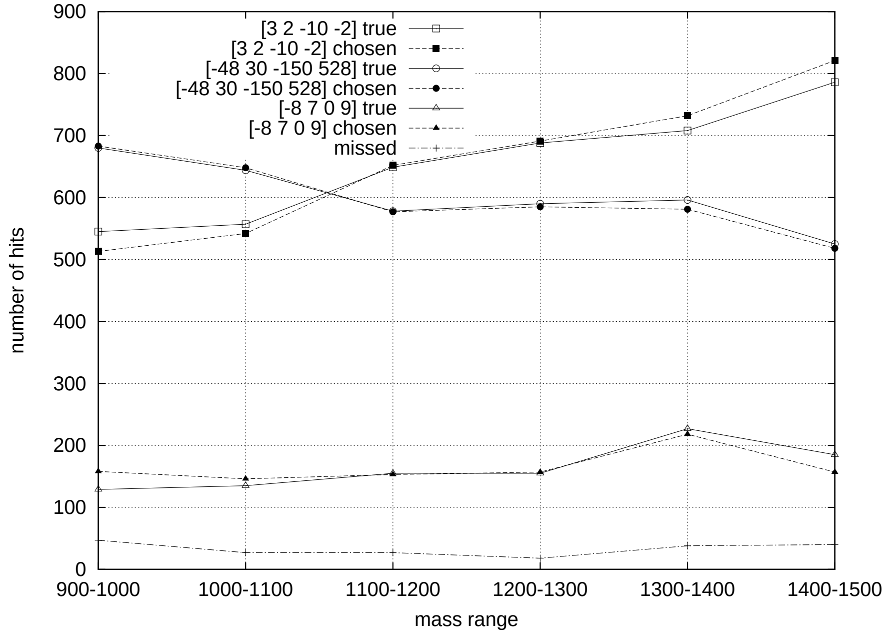

Fig. 7. Number of hits of a particular matrix chosen by our estimate (chosen), number of times when a particular matrix is actually the fastest (true) and the total number of wrong hits (missed). Numbers are counted over all peptides in the mass range in the simulated dataset.