Doctoral Thesis

Algorithms for Peptide Identification via Tandem Mass Spectrometry

Author(s): Tschager, Thomas

Publication Date: 2018

Permanent Link: https://doi.org/10.3929/ethz-b-000259599

Rights / License: In Copyright - Non-Commercial Use Permitted
Algorithms for Peptide Identification via Tandem Mass Spectrometry

A thesis submitted to attain the degree of
Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by

Thomas Tschager
MSc ETH UZH CBB
born on March 10, 1989
citizen of Italy

accepted on the recommendation of

Prof. Dr. Peter Widmayer examiner
Prof. Dr. Knut Reinert co-examiner

2018
Abstract

Proteomics is the systematic study of all proteins expressed in a cell. Mass spectrometry is able to measure the mass-to-charge ratio of a molecule and plays an important role in proteomics. Interpreting and understanding the data obtained by mass spectrometry requires methods developed in bioinformatics. In this thesis, we study algorithmic problems arising from data analysis of mass spectrometry-based proteomics.

In particular, we focus on the peptide sequencing problem, in which one wants to identify the sequence of amino acids a peptide is composed of from its mass spectrometric data. In a mass spectrometry experiment, multiple copies of a peptide are split at random positions into prefix and suffix fragments. Subsequently, the mass spectrometer measures the mass-to-charge ratios of these fragments and the fragment masses are deduced from these ratios. The amino acid sequence of the peptide has to be reconstructed from these masses. While some approaches try to find the sequence by searching in a database for a known sequence with similar fragment masses, other approaches reconstruct the sequence from scratch (de novo) by interpreting the fragment masses and their differences. We study problems in both de novo and database search approaches.

In the first part of this thesis, we study three de novo sequencing problems. We first propose a new scoring model for measuring the similarity of a set of fragment masses and a sequence of amino acids and develop a de novo sequencing algorithm for this model. Then, we study a problem variant that additionally exploits information obtained in an experimental step before the mass spectrometry anal-
ysis, where peptides in a complex mixture are separated by liquid chromatography. Finally, we consider the problem of reconstructing multiple sequences from a set of fragment masses that belong to different, simultaneously analyzed peptides. We develop an algorithm that reconstructs the sequences of all analyzed peptides at once.

In the context of database search approaches, we consider the blocked pattern matching problem. This problem is related to filtration techniques that are commonly used to speed up the database search. We consider the exact and two approximate variants of the problem that account for mutations, respectively post-translational modifications, and develop algorithms for all three variants.
Zusammenfassung


Im ersten Teil der Arbeit betrachten wir drei Probleme der De-Novo-Sequenzierung. Wir schlagen zuerst ein neues Modell zur

Im Bereich der Datenbanksuche betrachten wir das Blocked Pattern Matching Problem. Dieses Problem liegt Filtertechniken zugrunde, die in Datenbanksuchen häufig verwendet werden, um die Suche zu beschleunigen. Neben exakten Lösungen interessieren wir uns auch für zwei Variationen des Problems, bei denen Mutationen beziehungsweise posttranslationale Modifikationen erlaubt sind. Wir entwickeln Algorithmen für alle betrachteten Variationen des Problems.
Acknowledgments

Many people supported me while working on this thesis. First of all, I would like to thank Peter Widmayer for letting me be part of his group, supervising my doctoral studies, and providing such an encouraging working environment. I would like to thank Knut Reinert for taking the time to review my thesis and to examine me. My gratitude also goes to Tomas Hruz for his support and for always having time to discuss new ideas.

I would also like to thank my co-authors, colleagues and friends, particularly Andreas Bärtschi and Daniel Graf, and my family for their invaluable support.
# Contents

1 Introduction 1
   1.1 Identification via Mass Spectrometry . . . . . . . . 2
   1.2 Mathematical Model  . . . . . . . . . . . . . . . . . 3
   1.3 Overview and Summary of Results . . . . . . . . . 5

2 Background 9
   2.1 Peptides and Proteins  . . . . . . . . . . . . . . . . 9
   2.2 Instrumentation  . . . . . . . . . . . . . . . . . . . 13
      2.2.1 Liquid Chromatography  . . . . . . . . . . . 13
      2.2.2 Mass Spectrometry  . . . . . . . . . . . . . 14
   2.3 Experiments and Data Analysis  . . . . . . . . . . 15
      2.3.1 LC-MS/MS Experiments  . . . . . . . . . . . 16
      2.3.2 Peptide Sequencing  . . . . . . . . . . . . . 17
   2.4 Remarks on the Mathematical Model  . . . . . . . . 18

3 De Novo Sequencing 21
   3.1 Related Work  . . . . . . . . . . . . . . . . . . . . 23
   3.2 Symmetric Difference Scoring Model  . . . . . . . . 29
      3.2.1 Problem Definition  . . . . . . . . . . . . . 30
      3.2.2 Algorithm  . . . . . . . . . . . . . . . . . . 31
Chapter 1

Introduction

Mass spectrometry is a key technology in proteomics, the systematic study of all proteins that are expressed in a cell. Modern instruments are able to analyze complex mixtures of proteins with high throughput. Proteomics faces the problem of interpreting and understanding a huge number of mass spectra produced by such instruments. This data can usually only be understood with the help of computational methods developed in bioinformatics [93].

This thesis studies algorithmic problems in mass spectrometry-based proteomics. A fundamental task is the identification of the amino acid sequence a protein is composed of. All algorithmic problems we study arise from this task. In this chapter, we first introduce the basic concept of a tandem mass spectrometry experiment. We then describe our mathematical model for the identification problem and, eventually, give an overview of the results of this thesis.
1.1 Identification via Mass Spectrometry

A mass spectrometer measures the mass-to-charge ratios of molecules. Proteins are large molecules consisting of one or multiple peptides. A **peptide** is a chain of amino acids. In a typical (bottom-up) analysis, proteins are first cut into small peptides with ideally up to 20 amino acids. This digestion is not necessary, but often favorable, as it is difficult to analyze large peptides with mass spectrometers. The complex mixture of these small peptides is separated by liquid chromatography. In liquid chromatography, peptides need to traverse a column. The time for traversing the column is called *retention time* and depends on the interactions of the peptides with substances in the column. Ideally, each peptide has a distinct retention time. After this separation, the peptides are analyzed in a mass spectrometer.

Conceptually, a tandem mass spectrometry experiment can be divided into three phases. First, the mass spectrometer ionizes the peptide and measures its mass-to-charge ratio. Second, multiple copies of the same peptide molecule are fragmented at random positions into a prefix and a suffix fragment. Third, the instrument measures the mass-to-charge ratio of these fragments.

The analysis of the obtained data commonly starts with deducing the masses from the measured mass-to-charge ratios. Subsequently, the amino acid sequence of the peptide has to be reconstructed from the pattern of the fragment masses.

*Peptide sequencing* is the problem of determining the amino acid sequence of the peptide from its fragment masses. We cannot directly distinguish prefix and suffix fragment masses, which complicates the problem. In addition, fragmentation is a complex stochastic process
that is not completely understood and we do not measure each possible fragment mass. Finally, there exist various sources of errors in every step of the experiment.

Most approaches for peptide sequencing fall into one of the two following categories. First, one can compare the fragmentation pattern with the expected fragmentation pattern of all sequences in a database of known protein sequences and find the sequences with the most similar fragment masses. Second, approaches that are not limited to known protein sequences try to reconstruct the amino acid sequence of the peptide from scratch (de novo) by interpreting the fragment masses and their pairwise differences. In both cases, we need to score an amino acid sequence against a given fragmentation pattern.

1.2 Mathematical Model

We model amino acids by characters and peptides by strings throughout this thesis. Our model simplifies several aspects about peptide sequencing that we discuss at the end of Chapter 2 after describing proteins, the instrumentation, and the experimental design in more detail.

We consider a finite alphabet \( \Sigma \) and a function \( m : \Sigma \to \mathbb{N} \) that assigns each character \( a \in \Sigma \) its mass \( m(a) \in \mathbb{N} \). A peptide is a string \( S = a_1 a_2 \ldots a_n \) of characters in \( \Sigma \). The mass \( m(S) \) of \( S \) is the sum of the masses of its characters, \( m(S) := \sum_{i=1}^{n} m(a_i) \). We denote a substring of \( S \) by \( S_{i,j} := a_i \ldots a_j \) for \( 1 \leq i \leq j \leq n \). The empty string is denoted by \( S_\emptyset \) and has mass \( m(S_\emptyset) := 0 \).

In a tandem mass spectrometry experiment, we first measure the
mass $M$ of a peptide and the fragmentation pattern $X$, i.e. the set of some – but not necessarily all – of its fragment masses (Figure 1.1). The set $X$ can also contain masses that do not originate from fragments of the peptide (e.g. due to contamination or measurement errors). Throughout this thesis, we add the masses 0 and $M$ to $X$ to simplify the exposition.

Consider the example in Figure 1.1. The set $X$ does not contain all prefix and suffix masses of the analyzed peptide GAAMA. Especially, $X$ does neither contain the mass of the prefix G, nor of the complementary suffix AAMA. Therefore, we cannot decide from looking at $X$, whether the peptide is composed by the amino acids GAAMA or AGAMA. Moreover, $X$ contains the masses $x_3$ and $x_5$ that do not be-
long to fragments of the analyzed peptides. While there exist many approaches for filtering such masses in a preprocessing step, preprocessing cannot be assumed to remove all of these “foreign” masses.

We want to find a string with mass $M$ that explains the masses in $X$ as accurately as possible. To define accuracy, we need to compare a string with a set of fragment masses. We define the theoretical spectrum of a string $S$ as the set of all masses that we expect to measure in an ideal measurement of the peptide described by $S$. Formally, we define the set $\text{Pre}(S)$ of all prefixes of $S$ as $\text{Pre}(S) := \bigcup_{i=1}^{n} S_i \cup \{S_0\}$. The theoretical spectrum of $S$ is the union of all its prefix and suffix masses $\text{TS}(S) := \{m(T), m(S) - m(T) \mid T \in \text{Pre}(S)\}$. Note that for every prefix $T \in \text{Pre}(S)$ the string $S$ has a complementary suffix of mass $m(S) - m(T)$. The string $S$ explains a mass $m$ if $m \in \text{TS}(S)$.

A simple way for scoring a string $S$ against a fragmentation pattern $X$ is to count the number of masses in $X$ that are explained by $S$. Many variants of this scoring model have been proposed that take into account other information as well (e.g. the signal intensity reported by the instrument), or build a probabilistic scoring function based on this simple additive model.

1.3 Overview and Summary of Results

In Chapter 2, we describe important properties about proteins and the instrumentation, present the prevalent experimental designs, and discuss characteristics and simplifications of the mathematical model used in this thesis. The main results are organized as follows. We consider de novo sequencing problems in Chapter 3 and study a pattern matching problem for database search approaches in Chapter 4.
Symmetric difference scoring model In Section 3.2, we propose a new model for scoring a string $S$ against a fragmentation pattern $X$. Past approaches focused on computing a string that explains as many masses from $X$ as possible, i.e. to maximize the intersection of $\text{TS}(S)$ and $X$. We propose a scoring model that tries to explain masses in $X$, but also avoids to explain masses that are not in $X$. Formally, we are interested in a string $S$ that minimizes the symmetric difference between the theoretical spectrum of $S$ and the fragmentation pattern $X$. For this new optimization problem, we propose an efficient algorithm that computes both the best and the $k$ best solutions. This algorithm is able to handle more realistic fragmentation models that extend the theoretical spectrum of a string. These fragmentation models cannot be handled by many previous algorithms.

In an experimental evaluation, we compare the performance of the proposed symmetric difference scoring model with maximizing the size of the intersection of $\text{TS}(S)$ and $X$. The experiments show that the symmetric difference as optimization goal can improve the identification rates without substantial extra computational effort.

Exploiting additional information Algorithms for de novo peptide sequencing usually try to exploit as much information as possible from the mass spectrum of the peptide’s fragments. However, the information obtained from the separation by liquid chromatography preceding the mass spectrometric analysis is usually not considered. In Section 3.3, we formulate and study a new de novo sequencing problem that integrates liquid chromatography information as an additional constraint. We are interested in a string that both matches the fragmentation pattern and the retention time, i.e. the time a peptide needs for traversing the chromatographic column. We consider
three additive models for predicting the retention time of a peptide and develop algorithms for each model.

We compare the performance of our symmetric difference-based algorithms for two prediction models against the algorithm that minimizes the symmetric difference without considering the liquid chromatography information. The experiments suggest that identification rates are further improved by exploiting the chromatographic information.

**Sequencing from mixture spectra** Most algorithms for peptide sequencing are limited by the assumption that the fragmentation pattern $X$ is obtained by analyzing a single peptide. However, this assumption is not always fulfilled. In some types of experiments, the mass spectrometer selects all peptides within the same mass range for fragmentation and acquires mass spectra of the mixture of fragments of all selected peptides ($mixture$ $spectra$). In an iterative process, mixture spectra for all peptides in the same mass range are acquired every few seconds. In Section 3.4, we propose an algorithm for determining the sequences of $k$ simultaneously analyzed peptides given a set of mixture spectra acquired over some interval of time and the $k$ masses of the peptides. The algorithm uses a scoring function that does not consider a fragmentation pattern acquired at a specific point in time, but a series of consecutive mass spectra.

We implemented the proposed algorithm and study its performance on mixture spectra from the SWATH MS gold standard (SGS) dataset [80]. We show several examples, in which the results are improved by constructing the sequence of two peptides simultaneously instead of constructing only the sequence of a single peptide.

Chapter 4 addresses a problem arising from database search ap-
proaches in peptide sequencing. Known peptides can be reliably identified by comparing the fragmentation pattern with the theoretical spectrum of all sequences in a protein sequence database. To speed up this analysis, database filtration techniques quickly discard sequences with dissimilar fragment masses.

**Blocked pattern matching** A promising filtration technique uses a sequence of masses called blocked pattern to discard parts of a given database that are not relevant for the given query. A block refers to the mass of one or multiple characters. A blocked pattern with \( m \) masses matches a string if the string can be partitioned into \( m \) consecutive substrings such that the mass of the \( i \)-th substring is equal to the \( i \)-th mass in the pattern for all \( i = 1, \ldots, m \). The blocked pattern matching (BPM) problem is to find all strings in a database that match a given blocked pattern. In Section 4.3, we present a memory-efficient BPM algorithm that stores the database in a trie data structure and uses priority search trees to quickly traverse the trie. In Section 4.4, we consider two approximate BPM problem variants. The first variant allows for a mutation (insertion, deletion, or replacement) of a single character of the string matching the pattern. The second variant considers post-translational modifications and allows for an arbitrary number of modified characters in the string, where each character is modified by at most one modification of a predefined set of possible modifications. We present two algorithms for both variants of approximate blocked pattern matching.

An experimental evaluation shows that the proposed algorithm is fast in practice and that considering post-translational modifications considerably increases the number of identifiable spectra for several instrument types at a small computational cost.
Chapter 2

Background

In this chapter, we introduce some important aspects about proteins, the instrumentation used for their analysis, and common experimental designs. We refer to [86, 24] for a broader introduction in mass spectrometry-based proteomics and protein identification. In the last section of this chapter, we discuss some simplifying aspects of the mathematical model introduced in Section 1.2 and the underlying assumptions.

2.1 Peptides and Proteins

Proteins are large molecules that are composed by chains of amino acids and perform various different functions in the cell: from catalyzing reactions to regulating gene expression. The set of all pro-
Figure 2.1: Example of a peptide with three amino acids. All amino acids share a common structure and differ in their side chains ($R_1$, $R_2$, and $R_3$). The amino acids are linked by peptide bonds. The N-terminus of the peptide is the peptide’s end, at which the amino group is not part of a peptide bond. The other end is the C-terminus.

Proteins expressed in a specific cell at a specific time is called proteome\(^1\). Proteomics is the study of the proteome. In this section, we give a brief overview of the structure of proteins that are relevant for the problems considered in this thesis.

Proteins consist of one or multiple chains of amino acids. A chain of two or more amino acids is a peptide (Figure 2.1). Depending on their length, peptides are sometimes called oligopeptides (less than 20 amino acids) or polypeptides (having 20 or more amino acids). The twenty amino acids that occur in nature (Table 2.1) all share a common structure and differ in their so-called side chains. Amino acids are chained by the formation of peptide bonds (Figure 2.1). A water molecule is released during the formation of a peptide bond. Amino acids that are part of a peptide are called residue and the

\(^1\)There exist numerous other definitions of the proteome, e.g. as the set of all proteins encoded by the genome.
residue mass is the mass of the amino acid without the water molecule that is released when a peptide bond is formed. The residue masses of the 20 amino acids range between 57 and 186 dalton.

A peptide is directed in the sense that the two ends of the chain of amino acids can be distinguished. At one end, the N-terminus, is an amino group and at the other end, the C-terminus, is a carboxyl group. Amino acids are represented by single letters (Table 2.1) and the amino acid sequence is conventionally written from N-terminus to C-terminus. The mass of a peptide corresponds to the sum of the residue masses of its amino acids and, additionally, the mass of a water molecule, as the two amino acids at both ends of the peptide have only one peptide bond each.

Many proteins are modified after their synthetization. These post-translational modifications are important regulators of the protein’s function. Many post-translational modifications alter the chemical structure of a single amino acid and affect its – and hence the protein’s – mass. It can be of great interest to determine the exact location of such modifications for a deeper understanding of the protein’s function.
<table>
<thead>
<tr>
<th>Amino acid</th>
<th>3-letter code</th>
<th>1-letter code</th>
<th>Monoisotopic residue mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>Ala</td>
<td>A</td>
<td>71.04</td>
</tr>
<tr>
<td>Arginine</td>
<td>Arg</td>
<td>R</td>
<td>156.10</td>
</tr>
<tr>
<td>Asparagine</td>
<td>Asn</td>
<td>N</td>
<td>114.04</td>
</tr>
<tr>
<td>Aspartic acid</td>
<td>Asp</td>
<td>D</td>
<td>115.03</td>
</tr>
<tr>
<td>Asparagine</td>
<td>Cys</td>
<td>C</td>
<td>103.01</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Gln</td>
<td>Q</td>
<td>128.06</td>
</tr>
<tr>
<td>Glutamic acid</td>
<td>Glu</td>
<td>E</td>
<td>129.04</td>
</tr>
<tr>
<td>Glycine</td>
<td>Gly</td>
<td>G</td>
<td>57.02</td>
</tr>
<tr>
<td>Histidine</td>
<td>His</td>
<td>H</td>
<td>137.06</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>Ile</td>
<td>I</td>
<td>113.08</td>
</tr>
<tr>
<td>Leucine</td>
<td>Leu</td>
<td>L</td>
<td>113.08</td>
</tr>
<tr>
<td>Lysine</td>
<td>Lys</td>
<td>K</td>
<td>128.10</td>
</tr>
<tr>
<td>Methionine</td>
<td>Met</td>
<td>M</td>
<td>131.04</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>Phe</td>
<td>F</td>
<td>147.07</td>
</tr>
<tr>
<td>Proline</td>
<td>Pro</td>
<td>P</td>
<td>97.05</td>
</tr>
<tr>
<td>Serine</td>
<td>Ser</td>
<td>S</td>
<td>87.03</td>
</tr>
<tr>
<td>Threonine</td>
<td>Thr</td>
<td>T</td>
<td>101.05</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>Trp</td>
<td>W</td>
<td>186.08</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Tyr</td>
<td>Y</td>
<td>163.06</td>
</tr>
<tr>
<td>Valine</td>
<td>Val</td>
<td>V</td>
<td>99.07</td>
</tr>
</tbody>
</table>

Table 2.1: Amino acid codes and masses [24]. “Monoisotopic” means that the masses are calculated using the most abundant isotopes.
2.2 Instrumentation

2.2.1 Liquid Chromatography

Liquid chromatography (LC, [24, 71, 83]) is a widely used separation technique that is very important in mass spectrometry-based proteomics to analyze peptides in a complex mixture. In essence, the peptides in the mixture are dissolved in a liquid, the mobile phase, and move through a column containing a porous substance, the stationary phase.

The retention time of a peptide is the time the peptide needs to pass the column and depends on the peptide’s interaction with the mobile and the stationary phase. In this process, the peptides are usually separated by their hydrophobicity.

Liquid chromatography can be combined with mass spectrometry, such that peptides that elute one after another from the LC column are separately analyzed by the mass spectrometer. Separation by liquid chromatography is an important technique to reduce the complexity of a sample and to improve the sensitivity and the accuracy of the subsequent analysis.

Besides reducing the sample’s complexity, liquid chromatography can also aid in the subsequent data analysis steps. For example, the information obtained in the chromatographic separation can be used in peptide identification and quantification. Several studies try to predict the retention time of a peptide [4, 73]. This is a challenging task, as the retention time is affected by the type of the stationary and the mobile phase, the temperature, the dimensions and the type of the column, and other parameters of the experiments.
2.2.2 Mass Spectrometry

A mass spectrometer is an instrument for measuring the masses of molecules. More specifically, this instrument detects only charged molecules (ions) and thus peptides need to be charged (ionized) beforehand. Peptides are usually charged by adding protons. The charge state describes the number of added protons and is denoted by $z$. The mass spectrometer measures the mass-to-charge ratio ($m/z$) of ions. The output of a mass spectrometer, a mass spectrum, is the intensity of the recorded ions as a function of the mass-to-charge ratio. Converting mass-to-charge ratios to masses is often a necessary preprocessing step for the data analysis. Moreover, the signals of different isotopes of the same peptides is often merged to the monoisotopic mass of the peptide. Mass spectra contain valuable information for both the qualitative and quantitative analysis of complex mixtures of peptides. We refer to [24, 63, 1] for an introduction to mass spectrometry (MS) and its use in proteomics.

A mass spectrometer consists of three parts: an ionization source, a mass analyzer, and a detector. First, the ionization source ionizes the molecules to be analyzed. Then, the charged molecules are separated with respect to their mass-to-charge ratio by the mass analyzer. Finally, the molecules hit the detector, which records the signals. There exist different types of components for all three parts of the instrument that work with different principles and have different properties. These components can be combined in several ways.

Two or more stages of mass spectrometry (MS/MS) can be combined (either using two mass analyzers or two sequential analysis steps in the same analyzer). In the first stage, ions are selected with respect to their mass-to-charge ratio. Then, the selected ions are frag-
mented into charged fragments. In the second stage, a mass spectrum of the resulting charged fragments is measured. The fragmentation pattern recorded in this spectrum can then be used to characterize the amino acid sequence of the analyzed peptide. For the problems considered in this thesis, we will assume that the fragmentation pattern is preprocessed into a set of fragment masses.

Ideally, peptide fragmentation should only occur at the peptide bonds and result in two fragments, a prefix and a suffix fragment. An ideal fragmentation pattern should contain the masses of each possible prefix and suffix fragment of a peptide. Note that only charged fragments are detected by the instrument. In real experiments, a peptide does not only fragment at its peptide bonds, but also at other positions in the backbone, i.e. the parts of the amino acids that are common to all amino acids. Therefore, we need to distinguish different types of fragments that are called ion types. Fragmentation at bonds of the backbone other than the peptide bonds leads to small offsets in the masses of the resulting fragments that need to be considered when interpreting the fragmentation patterns. Besides prefix and suffix fragments, also other types of fragments are sometimes observed, e.g. internal fragments resulting from multiple fragmentation events at the backbone of the same peptide molecule, or fragments resulting from fragmentation at other bonds not belonging to the backbone. These types of fragments complicate the data analysis.

2.3 Experiments and Data Analysis

There exist two main types of experimental strategies: top-down and bottom-up analyses. In top-down analyses, whole proteins are
directly analyzed in the mass spectrometer, while in bottom-up analyses proteins are first digested into peptides. Mass spectrometry is then used to identify the amino acid sequences of these peptides and ultimately the proteins. Bottom-up analyses are the prevalent choice due to higher sensitivity, smaller measurement errors, and several other reasons [24].

2.3.1 LC-MS/MS Experiments

Liquid chromatography can be coupled with tandem mass spectrometry (LC-MS/MS) to analyze a complex mixture of peptides. The peptides elute one after another from the LC column and the mass spectrometer records a series of mass spectra over time in a cyclic procedure. In each cycle, the mass spectrometer first records a mass spectrum (MS$_1$ spectrum) of the charged peptides that elute from the LC column. Then, some peptides are selected for fragmentation using different strategies described below. Finally, one or multiple mass spectra (MS$_2$ spectra) of the charged fragments are acquired.

The selection of the peptides to be fragmented depends on the goal of the experiment [24]. In discovery proteomics, the mass spectrometer selects the $n$ highest signal intensities in the MS$_1$ spectrum for further fragmentation. Subsequently, $n$ MS$_2$ spectra are acquired, one for the fragments of each selected signal. In targeted proteomics, the aim is to identify specific peptides with high sensitivity. The scientist defines a list of m/z ratios and only signals with those m/z ratios are selected for fragmentation. In such analyses, also peptides with low abundance are analyzed. In some cases, an exclusion list of m/z values is defined as well, e.g. to avoid the selection of contaminants.
In contrast to this data-dependent acquisition strategies, a recent approach termed data-independent acquisition (DIA, [22]) aims for the analysis of all peptides in the sample. The whole m/z range of the MS1 spectrum is partitioned into intervals of a predefined width (e.g. 25 Dalton in SWATH MS [34]) and all intervals are considered one by one in a cyclic process. All peptides within an interval are simultaneously selected for fragmentation and a single MS2 spectrum of the fragments of all selected peptides is acquired. These spectra are called mixture spectra, as they contain signals of a mixture of fragments from many different peptides. In a DIA experiment, each peptide is selected for fragmentation at some point and the whole sample is analyzed. However, the data analysis requires special software designed for the interpretation of mixture spectra.

2.3.2 Peptide Sequencing

Peptide sequencing is the problem of characterizing the amino acid sequence of a peptide from the data acquired in a LC-MS/MS experiment. Most approaches can be distinguished by whether they make use of a database of known protein sequences or whether they construct a sequence matching the given measurement data from scratch. The former class comprises algorithms that try to find one or multiple sequences in a database that match a given MS2 spectrum in the best possible way. On the other hand, the latter is the class of so-called de novo peptide sequencing algorithms that try to reconstruct the amino acid sequence of a peptide by interpreting only the signals in the MS2 spectrum mainly considering the mass differences of the peptide fragments.

There exist also approaches that try to combine database search
with de novo sequencing. In many such approaches, de novo sequencing is used as a database filtration step in order to speed up the subsequent database search. Unfortunately, such approaches suffer from the principle limitation that the search space is limited to the database of known sequences.

Whether we study database search approaches or de novo sequencing algorithms, in both cases we need to define a scoring function for comparing an amino acid sequence with a MS$_2$ spectrum of fragment masses. Such scoring functions rely on a model of peptide fragmentation that is able to predict the expected occurrence of a fragment and ideally also the corresponding signal intensity in the MS$_2$ spectrum. Unfortunately, the fragmentation of peptides is not fully understood. Models for peptide fragmentation can be distinguished into statistical approaches that learn rules from large sets of annotated MS$_2$ spectra and chemical approaches that try to understand the exact chemical processes in the fragmentation step. We refer to [24, Chapter 9] for an introduction in models for peptide fragmentation.

### 2.4 Remarks on the Mathematical Model

A peptide is not only composed by the chain of amino acid residues, but additionally of an oxygen and two hydrogen atoms. Therefore, the mass of an uncharged peptide is the sum of its amino acid residue masses and the mass of the additional H$_2$O molecule (18 Dalton).

To simplify the description of the algorithms, we do not consider the H$_2$O mass and deal with the mass $M$ that is the sum of the amino acid residue masses of the peptide, where the H$_2$O mass has already been subtracted. For the same reason, we add both 0 and the mass $M$
to the fragmentation pattern \( X \). Similarly, we do not consider the fragment mass offsets of different ion types in our definition of the theoretical spectrum. We will discuss how our de novo sequencing algorithms can consider these offsets. The implementations of our de novo sequencing algorithms support a range of ion types that is larger than supported by several previous approaches.

Note that our model considers fragment masses, but that the mass spectrometer measures masses-to-charge ratios. Charge state deconvolution \([49]\) is required as a preparatory step to convert mass-to-charge ratios to masses if multiply charged fragments should be considered.

While we do not explicitly model post-translational modifications, our model can consider both fixed and variable modifications. Fixed modifications occur at every amino acid residue in a sequence and can be considered by altering the amino acid masses accordingly. Variable modifications occur only at some residues and can be supported by adding new characters to the alphabet, such as proposed in \([42]\).

Finally, we consider the masses to be integer values in the description of the algorithm and ignore the accuracy of the mass spectrometer. We can account for the accuracy of the instrument by multiplying the masses by an appropriate factor before rounding to integers. Additionally, in the implementation of our algorithm we consider masses to be equal if they differ at most by a predefined error tolerance.
Chapter 3

De Novo Sequencing

De novo peptide sequencing has been studied for more than 30 years and a wide range of approaches to tackle this problem have been proposed. In this chapter, we first try to give an overview over the prevalent algorithmic techniques and their limitations in Section 3.1. We refer to [2, 65, 59] for more comprehensive surveys of de novo sequencing and to [68] for a recent comparison of popular de novo sequencing tools.

In Section 3.2, we propose a new scoring model, namely to consider the symmetric difference between the fragmentation pattern $X$ and the set of masses that a string explains. We study the problem of minimizing the size of the symmetric difference and develop an algorithm that computes both the best and the $k$ best solutions.

In Section 3.3, we study the problem of computing a sequence that is not only in accordance with the fragmentation pattern $X$, but
also with the retention time observed in the liquid chromatography experiment. We consider three models for predicting the retention time of a string and develop algorithms for de novo sequencing with respect to these models.

Finally, we consider de novo peptide sequencing for mixture spectra in Section 3.4. We propose an algorithm that reconstructs the sequences of multiple, simultaneously measured peptides and develop a scoring function that accounts for multiple spectra acquired over time.

At the end of each section, we evaluate the developed algorithms on real experimental data. We consider data from the SWATH MS gold standard dataset [86], an annotated and manually validated dataset measured in data-dependent and data-independent acquisition mode. In this dataset, 422 synthesized peptides are measured in various concentrations and in mixtures of different complexities, namely without additional peptides, in yeast proteomic background and in human proteomic background.
3.1 Related Work

**Brute-force** The first, simple approaches for de novo peptide sequencing separated the task of generating candidate strings from the problem of scoring strings against a fragmentation pattern [40, 81]. For the candidate string generation, these algorithms use a simple brute-force approach and generate all strings with the given mass $M$. These candidate strings are then scored using a simple scoring function that essentially counts the number of masses in $X$ that a string explains. However, exhaustively scoring many candidate strings generated in this way is only suitable for very small and very accurately measured masses.

**Graph-based** Many algorithms [15, 8, 57, 5, 37] for de novo peptide sequencing use a graph-based approach to restrict the search space of candidate strings. The idea of this approach is to model the fragmentation pattern $X$ as a directed acyclic graph, the *spectrum graph* [7]. The vertices in this graph are masses. Two vertices are connected by an edge if there exists a string with a mass equal to the mass difference of the incident vertices. Edges are always directed from the smaller to the larger mass. A path from vertex 0 to vertex $M$ represents a set of strings with mass $M$ that all share a common set of prefix masses containing every traversed vertex. In many cases, the set $X$ does not contain the mass of every prefix fragment. As we have redundant information (every fragmentation event leads to two complementary fragments), we can sometimes recover a missing mass $m$ by adding the complementary mass $M - m$ to the vertex set of the spectrum graph. In practice, the size of the edge set is often reduced by limiting the maximal considered mass difference or the number of characters.
a string matching the mass difference can have.

Instead of generating strings with a given mass, such approaches then try to find a path from 0 to $M$ with optimal score. Various additive scoring functions have been proposed that either define a score for individual edges or individual vertices. A straight-forward dynamic program [13] can compute the longest (or highest-scoring) path in a directed acyclic graph. However, algorithms that compute the longest path in the spectrum graph do not rigorously score the complementary suffix fragments. This is due to the fact that we cannot distinguish if a mass in $X$ is the mass of a prefix or of a suffix fragment. If a path uses a vertex, it is interpreted as prefix fragment mass and the complementary vertex as suffix fragment mass. Depending on the scoring function, it can be very likely that many pairs of complementary vertices are traversed by a path. These symmetric paths rarely represent correct solutions and are rather an artifact of oversimplification.

Chen et al. [8] formulate a more adequate path finding problem, where such double-interpretations of complementary vertices are forbidden: A path in the spectrum graph is called antisymmetric if it does not traverse any pair of complementary vertices. De novo sequencing is reduced to the problem of finding the longest antisymmetric path in the spectrum graph. In a different context, this problem has been formulated as the longest path avoiding forbidden pairs problem [33], i.e. the problem of finding a longest path using at most one vertex of every given forbidden pair of vertices. In this formulation, the forbidden pairs in the spectrum graph correspond to the pairs of complementary vertices. This problem has been shown to be NP-hard in general [33, 51], but can be solved in polynomial time for forbidden pairs with some specific structural properties. In
spectrum graphs, forbidden pairs are always nested, i.e. for every pair of complementary vertices \((m, M-m)\) and \((m', M-m')\) it holds that either \(m \leq m'\) and \(M - m' \leq M - m\) or that \(m' \leq m\) and \(M - m \leq M - m'\). For nested forbidden pairs, the longest path avoiding forbidden pairs problem can be solved in polynomial time [8]. An algorithm that computes the \(k\) best-scoring paths in a spectrum graph has been proposed in [57] and several other algorithms are based on the algorithm of Chen et al. [8], for example [30].

However, the limitation to a nested structure of forbidden pairs restricts the possibilities in modeling additional aspects of de novo sequencing. For example, it is not possible to account for multiple charge states or all possible ion types with nested forbidden pairs. Bafna and Edwards [5] define simple ion types with a restricted mass offset from the corresponding prefix or suffix fragment mass and develop a dynamic programming algorithm accounting for such ion types using the same algorithmic tools as in [8]. In Section 3.2, we present a different algorithm that is able to consider a broader set of possible ion types. Interestingly, some recent approaches [45, 58] for de novo sequencing do not consider forbidden pairs and try to avoid symmetric solutions by designing better scoring functions.

**Other approaches** Another dynamic programming approach [61], that has been integrated in the popular commercial de novo sequencing software PEAKS [60], uses the fragmentation pattern \(X\) only for scoring, but not to restrict the search space. While this approach is not explicitly graph-based, it exploits the same nested structure of forbidden pairs as the algorithm for the longest antisymmetric path problem [8].

In [29], the fragmentation process of peptides is described by a
hidden Markov model. This approach faces the same problem of not being able to distinguish prefix and suffix masses that leads to the introduction of the antisymmetry constraint in graph-based approaches. To reduce the computational complexity, the complete hidden Markov model is approximated by separating the problem of distinguishing prefix and suffix masses and the problem of constructing the actual string (i.e. picking the masses of $X$ that are explained by the solution).

An integer linear programming formulation of the longest anti-symmetric path problem has been proposed in [21, 3]. This formulation is flexible and not restricted to nested structures of forbidden pairs. For example, the formulation does not only allow for considering all possible types of ions, but also allows for more global constraints (e.g. restricting the number of specific amino acids).

**Scoring models** Almost all scoring functions only consider prefix and suffix fragment masses and do not allow for masses to be interpreted as internal fragment. In fact, the de novo sequencing problem with internal fragments has been shown to be NP-complete [95]. Many scoring functions – especially those designed for graph-based algorithms – are additive and essentially score each explained mass, respectively pair of complementary masses, individually. While simple examples of such scoring functions either count the number of explained masses (shared peaks count) or sum up the corresponding signal intensities [61], different probabilistic approaches have been developed to account for dependencies between fragments originating from the same fragmentation event [30, 29].

Some scoring functions take into account the flanking amino acids when modeling the probability of a fragmentation event. Such
functions can be considered by an algorithm [37] similar to [8] that replaces a vertex in the spectrum graph by multiple copies, one for each incident edge. More general scoring functions consider global aspects of the fragmentation process. These scoring functions are usually not integrated in graph-based algorithms, but used in combination with a simpler, additive scoring function and a fast algorithm for generating candidate strings. The candidate strings are then scored using the general scoring function [58].

**Mixture spectra** Most de novo sequencing algorithms are limited by the assumption that the given mass spectrum is obtained from a single peptide and does not contain masses of other peptides’ fragments. However, for peptides with similar masses, that are not well separated by liquid chromatography, it can happen that a spectrum of the fragments of all of these peptides is acquired. Such spectra containing the fragment masses of multiple peptides are called mixture spectra. In data-independent acquisition (DIA) experiments, mixture spectra are rather the normal case than the exception due to the experimental design.

A common strategy for peptide identification from mixture spectra [94, 36, 92, 72] is to preprocess the spectra in order to distinguish fragment masses of different peptides in the mixture. This preprocessing is helpful, as peptide identification rates of single-peptide algorithms are harmed by mixture spectra [36, 43]. The signals of different peptides are distinguished using their chromatographic similarities, i.e. by the similarity of the signal intensities over time. The result is a set of pseudo-spectra that are supposed to contain only signals of a single peptide and can be analyzed with traditional peptide identification algorithms. However, this preprocessing strongly
relies on sufficiently different signal intensities of co-fragmented peptides. In cases, where peptides cannot be distinguished by their chromatographic features, traditional identification tools (both database search and de novo approaches) cannot be used.

There has been only little research on de novo sequencing for mixture spectra. An algorithm for the de novo sequencing problem for two peptides is proposed in [56] and extends the approach of [61]. This research is motivated by mixture spectra from data-dependent acquisition measurements. In Section 3.4, we discuss an algorithm for de novo peptide sequencing of $k$ peptides from mixture spectra obtained in data-independent acquisition mode.
3.2 Symmetric Difference Scoring Model

In this section, we develop an algorithm for de novo peptide sequencing with the symmetric difference scoring model. Several approaches [8, 61] aim for a string $S$ with mass $M$, such that the set $TS(S)$ of all prefix and suffix masses of $S$ contains as many masses as possible of the given fragmentation pattern $X$. This scoring model is often referred to as *shared peaks count*. Besides only considering the size of the intersection $TS(S) \cap X$, several of these approaches [15, 45, 66] can also maximize a more elaborate score on the masses in $TS(S) \cap X$.

However, considering only the intersection of $TS(S)$ and $X$ might lead to a bias towards the use of amino acids with small masses. For example, the amino acid glutamine has the same mass as the sum of the masses of a glycine and an alanine. When maximizing $|TS(S) \cap X|$, one can always replace a glutamine by both a glycine and an alanine in the string $S$ without decreasing the size of the intersection. In an ideal experiment, where all prefix and suffix masses and no other masses are given in $X$, there exists a string $S$ with $TS(S) = X$. However, in a real-world experiment with missing masses, we want to explain masses that are in $X$, but not to explain masses that are not in $X$. Dančík et al. [15] noted this problem and proposed a probabilistic scoring model incorporating penalty scores for some specific fragment masses present in $TS(S)$ but not in $X$. However, current algorithms do not systematically account for exactly those masses in $TS(S) \setminus X$.

We propose a scoring model that considers both the masses in $TS(S) \cap X$ and the masses in $TS(S) \setminus X$. Conceptually, our aim is
to minimize the size of the symmetric difference $|TS(S) \triangle X| = |TS(S) \setminus X| + |X \setminus TS(S)|$ instead of maximizing the size of the intersection $|TS(S) \cap X|$. We explore this scoring model by first giving a precise definition of our new optimization problem and by developing an algorithm for this problem. Then, we provide a proof-of-concept implementation and study how the symmetric difference improves over the shared peaks count in terms of quality of the result. Our experiments demonstrate that the symmetric difference scoring model leads to higher identification rates that do not come at an unbearable computational cost.

3.2.1 Problem Definition

We want to find a string $S$ that explains a given fragmentation pattern $X$ as accurately as possible. We define the score of a string $S$ and a fragmentation pattern $X$ as an additive function

$$\text{score}(S, X) = \sum_{m \in TS(S)} f_*(m, X), \quad (3.1)$$

where $f_*(m, X) \in \mathbb{R}$ indicates the score of a mass $m$ that is explained by $S$ depending on whether $m$ is in $X$ or not. Past approaches [8, 61] often considered the shared peaks count, where one uses

$$f_{scp}(m, X) = |\{m\} \cap X| = \begin{cases} 1 & \text{if } m \in X, \\ 0 & \text{if } m \notin X, \end{cases} \quad (3.2)$$

and variants of it. Conceptually, the shared peaks count computes the number of masses that are both in $TS(S)$ and $X$. 

30
We aim to minimize the symmetric difference between TS(S) and X. Equivalently, we can solve the problem of finding a string S that maximizes |TS(S) ∩ X| − |TS(S) \ X|. The reason for this observation is that for a fixed X, a chosen S that maximizes the latter also minimizes the symmetric difference. Hence, we can define

\[
f_\triangle(m, X) = |\{m\} \cap X| - |\{m\} \\setminus X| = \begin{cases} 
1 & \text{if } m \in X, \\
-1 & \text{if } m \notin X.
\end{cases}
\]  

We formulate the de novo peptide sequencing with the symmetric difference scoring model as follows.

**Problem 1** \((\text{DeNovoSymDiff})\). Let \(\Sigma\) be an alphabet of characters with a mass \(m(a) \in \mathbb{N}\) for each \(a \in \Sigma\). Given the peptide mass \(M \in \mathbb{N}\) and a set \(X = \{x_i \in \mathbb{N} \mid i = 1, \ldots, k\}\) of fragment masses, find a string \(S\) of characters in \(\Sigma\) with \(m(S) = M\) that maximizes

\[
\text{score}(S, X) = \sum_{m \in \text{TS}(S)} f_\triangle(m, X).
\]

**3.2.2 Algorithm**

In this section, we present an algorithm for solving the problem DeNovoSymDiff. Our algorithm builds on the seminal graph-based algorithm of Chen et al. [8] that computes a string that maximizes the shared peaks count.

The search space of our algorithm \(\text{DeNovo}_\triangle\) is modeled by a directed acyclic multigraph \(G = (V, E)\) based on the fragmentation pattern \(X\). A vertex in \(G\) represents a mass and a path in \(G\) represents
Figure 3.1: The set $X$ is depicted on the real line (bottom). Masses in $X$ are denoted by vertical bars and masses $m$ with $m \not\in X$ and $M - m \in X$ by crosses. A subgraph of our multigraph is depicted in the upper part. Only edges with label of length one and no parallel edges connecting the same set of vertices are shown.

A string. For every mass $m \in X$ there are two vertices $m$ and $M - m$ in $G$, i.e. $V = \{m, M - m \mid m \in X\}$. An edge in $G$ is always directed from the smaller to the larger mass. Two vertices $v$ and $w$ are connected by an edge if there exists a string with mass $w - v$. For each such string with mass $w - v$, we add an edge from $v$ to $w$ to the multigraph and label it with this string. That is, if $v$ and $w$ are connected by an edge with label $l(v, w)$, there is also an edge from $v$ to $w$ for every permutation of $l(v, w)$. In practice, we only consider edges with small labels (up to two or three characters). An example of this multigraph is depicted in Figure 3.1. For simplicity, only edges connecting two vertices with a mass difference equal to the mass of a single character are shown.

A path from $v$ to $w$ represents a string of mass $w - v$. We denote the concatenation of the edge labels of a path $P$ by $l(P)$. Let $P = (0, v_1, \ldots, v_k, M)$ be a path from vertex 0 to vertex $M$. Every tra-
versed vertex $v_i$ represents the mass of a prefix of the string $l(P)$. The path with label $l(P)$ both explains $v_i$ and $M - v_i$ for every traversed vertex $v_i$.

In the example (Figure 3.1), two strings that maximize the number of explained masses in $X$ are AGFSGQV or AGQYQV. While the first string explains masses that are not in $X$ (crosses), all explained masses of the second string are in $X$ (Figure 3.2). We feel, that the second string is more likely the correct string and therefore aim for minimizing the symmetric difference.

The idea of DeNovo$\Delta$ for dealing with the symmetry in the spectrum graph (traversing a vertex $m$ also explains $M - m$) is to iteratively extend two paths both starting at vertex 0. One path represents a prefix and the other path a reversed suffix of $S$. DeNovo$\Delta$ extends these paths until they end in two vertices $v$, respectively $M - v$. In other words, the algorithm extends both paths until the sum of their labels’ masses is equal to $M$. Finally, the prefix and the reversed suffix string are concatenated to a string of mass $M$.

In every step, the algorithm extends the path that represents the
substring of smaller mass. We call this procedure \textit{balanced extension}.

\textbf{Definition 1 (Balanced extension).} \textit{Given two paths }$P$\textit{ and }$Q$\textit{ both starting at vertex }$0$, a balanced extension extends the path that represents the string of smaller mass by a single edge, unless the resulting paths represent strings with a total mass larger than }$M$. \textit{An arbitrary path is extended if both paths represent strings with equal masses.}

By always performing balanced extensions we assure that the two paths represent strings with similar masses throughout the execution of the algorithm. We call two paths that result from a sequence of balanced extensions a \textit{path pair}.

\textbf{Definition 2 (Path pair).} A path pair is a pair of paths $P = (0, \ldots , v)$ and $Q = (0, \ldots , a, b)$ in $G$ that results from a sequence of balanced extensions starting from two paths $P_0 = (0)$ and $Q_0 = (0)$.

Let $P = (0, p_1, v)$ and $Q = (0, q_1, a, b)$ be a path pair with $v \leq b$ and $v + b \leq M$ (Figure 3.3). We know that $a \leq v$, as the algorithm always extends the path ending in the smaller mass. If $a$ would be larger than $v$, the algorithm would not have extended the subpath of $Q$ ending in $a$ by the edge $(a, b)$ in a previous step, but $P$ instead (by some other edge). Based on this observation, \textsc{DeNovo$\Delta$} can efficiently update the number of explained masses that are in $X$, respectively not in $X$, while extending the paths.

The set of masses that are explained by a path pair $(P, Q)$ is the \textit{partial theoretical spectrum}

$$\text{PTS}(P, Q, M) := \{ m(T), M - m(T) \mid T \in \text{Pre}(l(P)) \cup \text{Pre}(l(Q)) \}.$$  

(3.4)
Figure 3.3: The paths $P = (0, p_1, v)$ and $Q = (0, q_1, a, b)$ are a path pair, as there exists a sequence of balanced extensions resulting in these two paths. A balanced extension by $(v, w)$ leads to a path pair $(P', Q)$, with $P' = (0, p_1, v, w)$ and $m(l(P')) + m(l(Q)) = M$. The path labels of these to paths are then combined to the string $AGSGIDIK$ with mass $M$. Dotted lines point towards masses on the real line (bottom) that are in $TSe((a, b), M)$, respectively $TSe((v, w), M)$ (the complementary masses are omitted). As shown in this example, these two sets can have a non-empty intersection.

Note that every mass in the partial theoretical spectrum of the path pair $(P, Q)$ with $a \leq v \leq b$ is either at most as large as $b$ or larger or equal $M - b$.

Assume that the algorithm extends $P$ by an edge $(v, w)$ in the next step (dashed edge in Figure 3.3). By this extension, we explain the following set of additional masses

$$TSe((v, w), M) := \{ m(T) + v, M - (m(T) + v) \mid T \in \text{Pre}(l(v, w)), m(T) \neq 0 \}. \quad (3.5)$$

Note that we do not consider the empty prefix in $\text{Pre}(l(v, w))$, be-
cause \( v \) and \( M - v \) are already explained by \( P \). Every mass in \( \text{TSe}((v, w), M) \) is larger than \( v \) and smaller than \( M - v \). If the edge \((v, w)\) would be labeled by a single character, \( \text{TSe}((v, w), M) \) would contain only two masses, namely \( w \) and \( M - w \).

**Lemma 1.** For every path pair \( P = (0, \ldots, v) \) and \( Q = (0, \ldots, a, b) \) with \( v \leq b \) and \( v + b \leq M \) it holds that \( a \leq v \leq b \). The balanced extension of \((P, Q)\) by an edge \((v, w)\) additionally explains all masses in \( N((v, w), (a, b)) := \text{TSe}((v, w), M) \setminus \text{TSe}((a, b), M) \).

**Proof.** Assume that there exists a path pair \((P, Q)\) with \( v \leq a \). This path pair results by definition from a sequence of balanced extensions. Consider the balanced extension in this sequence, where the last edge \((a, b)\) of \( Q \) is added. In this step, either \( P \) ended in \( v \) or in some vertex \( v' < v \). In both cases, \( a \) is the larger mass and \( Q \) represents the heavier string. Hence, the extension by \((a, b)\) is not a balanced extension and \((P, Q)\) is not a path pair.

Consider a balanced extension of \((P, Q)\) by an edge \((v, w)\). By this extension, we explain all masses in \( \text{TSe}((v, w), M) \). However, some of these masses might also be explained by \((P, Q)\). We show that \( \text{TSe}((v, w), M) \setminus \text{PTS}(P, Q, M) \) is exactly \( N((v, w), (a, b)) \). In other words, we show that all masses explained by \((v, w)\) that are also explained by \((P, Q)\), are explained by the last edge \((a, b)\) of \( Q \). We note that all masses in \( \text{TSe}((v, w), M) \) are larger than \( v \) and smaller than \( M - v \). Moreover, all masses in \( \text{PTS}(P, Q, M) \) that are larger than \( v \) and smaller than \( M - v \) are explained by the edge \((a, b)\). Therefore, it follows that the balanced extension with \((v, w)\) additionally explains all masses in \( N((v, w), (a, b)) \) \( \Box \).

By Lemma 1, the following invariant holds for every path pair
(P, Q) with P = (0, ..., v) and Q = (0, ..., a, b): All masses, that are both explained by some outgoing edge (v, w) of v and by P or Q, are in TSe((a, b), M). This invariant holds for any path pair computed by the algorithm (even if the paths share no explained masses) and any edge that is feasible for a balanced extension.

Thus, the algorithm does not have to remember all vertices of a path pair to compute the newly explained masses after an extension. It is sufficient to remember the last two vertices of each of the paths, namely v and w, respectively a and b. By Lemma 1, the additional score of such a balanced extension can be defined as

$$\text{gain}((v, w), (a, b)) := \sum_{m \in N((v, w), (a, b))} f_\Delta(m, X).$$

We compute a string with mass M that minimizes the symmetric difference a with dynamic program. We define a two-dimensional table T with |V| rows and |E| columns, where V denotes the set of vertices and E the multiset of edges of G. An entry $T[v, (a, b)]$ contains the optimal score of any two paths $P = (0, \ldots, v)$ and $Q = (0, \ldots, a, b)$, i.e.

$$T[v, (a, b)] = \max_{P, Q} \left\{ \sum_{m \in \text{PTS}(P, Q, M)} f_\Delta(m, X) \right\},$$

where the maximum is taken over all path pairs with one path ending in the vertex v and the other path in the edge (a, b). We only consider an entry $T[v, (a, b)]$ if $a \leq v \leq b$ and $v + b \leq M$.

With the invariant described above, we can compute the value of $T[v, (a, b)]$ given the values of all entries $T[x, (c, d)]$ with $x < v$ or $x = v$ and $c < a$ as follows (Figure 3.4). Let $P = (0, \ldots, v)$ and
Figure 3.4: Computation of $T[v,(a,b)]$: Either the path $P = (0, \ldots, v)$ ends in an edge $(v', v)$ with $v' \leq a$ (left) or it ends in an edge $(v'', v)$ with $v'' > a$ (right). The dashed edge is the edge added in the last balanced extension leading to $(P, Q)$.

Let $Q = (0, \ldots, a, b)$ with $a \leq v \leq b$ be the two paths that maximize the score among all path pairs ending in $v$ and $(a, b)$. We consider all incoming edges of $v$ and distinguish two cases. Either the last edge of $P$ starts at a source vertex that is at most as large as $a$ or at a source vertex that is larger than $a$.

In the former case, a subpath of $Q$ was extended by the edge $(a, b)$ in the last extension step before reaching $P$ and $Q$. Hence, for an edge $(v', v)$ with $v' \leq a$, we consider the value of $T[a,(v', v)]$, that is the maximum score of any two paths ending in $a$ respectively $(v', v)$, and add the additional score of $(a, b)$, i.e. $\text{gain}((a, b), (v', v))$.

In the latter case, a subpath of $P$ was extended by an edge ending in $v$ in the last step before reaching $P$ and $Q$. For an edge $(v'', v)$ with $v'' > a$, we add $\text{gain}((v'', v), (a, b))$ to the value of $T[v'', (a, b)]$.

We consider all incoming edges of $v$ in this way in order to cover all possibilities for reaching $P$ and $Q$. Hence,
\[
T[v,(a,b)] = \max \left\{ \begin{array}{l}
\max_{\nu' \leq a} \left\{ T[a, (\nu', \nu)] + \text{gain}((a, b), (\nu', \nu)) \right\} \\
\max_{\nu'' > a} \left\{ T[\nu'', (a, b)] + \text{gain}((\nu'', \nu), (a, b)) \right\}
\end{array} \right. 
\]

(3.8)

In the pseudocode of DeNovo$\Delta$ (Algorithm 1), we use a formulation that is slightly different from Equation (3.8), as it simplifies the analysis of the algorithms’ time complexity. The algorithm first initializes every entry of table $T$ by $-\infty$. To simplify the notation, we assume that $E$ contains a loop edge $(0,0)$ and set $T[0,(0,0)] = 2$ (the path pair representing two empty strings explains the masses $0,M \in X$). Then, the algorithm considers all vertices $v$ in ascending order and for a vertex $v$ all edges $(a,b)$ (with $T[v,(a,b)] \neq -\infty$) in ascending order of $a$ and $b$. It extends the path ending in $v$ by every outgoing edge of $v$ and updates the corresponding entry in $T$. Once all entries have been computed, the optimal score for a string of mass $M$ is equal to the maximum value of an entry $T[M-b,(a,b)]$ among all edges $(a,b)$ in $G$.

**Theorem 1.** Given a peptide mass $M \in \mathbb{N}$ and a fragmentation pattern $X = \{x_i \in \mathbb{N} \mid i = 1, \ldots, k\}$, algorithm DeNovo$\Delta$ computes a solution for DeNovoSymDiff.

**Proof.** We prove by induction that algorithm DeNovo$\Delta$ computes the entries of table $T$ correctly. As base case, we see that the entries
\textbf{Algorithm 1: DeNovo$\Delta$}

\begin{align*}
T[v,(a,b)] &\leftarrow -\infty \text{ for all } (a,b) \in E \text{ and } v \in V \\
T[0,(0,0)] &\leftarrow 2 \\
\textbf{foreach } v \in V \text{ in ascending order do} \\
\quad \textbf{foreach } (a,b) \in E \text{ in lexicographically ascending order do} \\
\quad \quad \textbf{foreach } (v,w) \in E \text{ with } w + b \leq M \text{ do} \\
\quad \quad \quad \text{new} &\leftarrow T[v,(a,b)] + \text{gain}((v,w),(a,b)) \\
\quad \quad \quad \textbf{if } w \leq b \text{ then} \\
\quad \quad \quad \quad T[w,(a,b)] &\leftarrow \max(T[w,(a,b)], \text{new}) \\
\quad \quad \quad \textbf{else} \\
\quad \quad \quad \quad T[b,(v,w)] &\leftarrow \max(T[b,(v,w)], \text{new})
\end{align*}

$T[0,(0,v)]$ for all $(0,v) \in E$ are computed correctly. Assume that all entries $T[v',(a',b')]$ with $v' < v$ or $a' \leq v' = v$ are correct. The next entry $T[v,(a,b)]$ is either computed using an entry $T[a,(v',v)]$ with $v' \leq a$ or an entry $T[v'',(a,b)]$ with $a < v''$. Both entries are correct by the induction hypothesis. In the first case, $T[a,(v',v)] = \sum_{m \in \text{PTS}(P',Q,M)} f_{\Delta}(m,X)$ for some paths $P' = (0,\ldots,a)$ and $Q = (0,\ldots,v',v)$. A path $P$ ending in $b$ can be constructed by extending $P'$ with the edge $(a,b)$. We show that

\begin{align*}
T[v,(a,b)] &= \sum_{m \in \text{PTS}(P',Q,M)} f_{\Delta}(m,X) + \text{gain}((a,b),(v',v)) \\
&= \sum_{m \in \text{PTS}(P,Q,M)} f_{\Delta}(m,X).
\end{align*}
By Lemma 1, no mass considered by \( \text{gain}((a, b), (v', v)) \) has already been accounted for when computing the score of \((P', Q)\). Similarly, in the second case the masses considered by \( \text{gain}((v'', v), (a, b)) \) have not been considered when computing the score of \( T[v'', (a, b)] \).

Finally, we show that any string with an optimal score can be constructed by balanced extensions. Let \( S \) be an optimal string. There are exactly two consecutive prefixes of \( S \) with masses \( v \) and \( w \) such that \( v \leq M/2 < w \). The entry \( T[M - w, (v, w)] \) is equal to

\[
\sum_{m \in \text{PTS}(P, Q, M)} f_\triangle(m, X)
\]

for some paths \( P = (0, \ldots, M - w) \) and \( Q = (0, \ldots, w) \). Concatenating \( l(P) \) and the reversed string of \( l(Q) \) either results in \( S \) or in another string \( S' \) with the same, optimal score.

**Theorem 2.** The time complexity of algorithm \( \text{DeNovo}_\triangle \) is in \( O(|V| \cdot |E| \cdot d \cdot p) \), where \( d \) is the maximal out-degree of a vertex in \( G \) and \( p \) is the maximal length of an edge label.

**Proof.** The table \( T \) can be initialized in \( O(|V| \cdot |E|) \) time. To compute an entry \( T[v, (a, b)] \), the algorithm considers all outgoing edges of \( v \), that is at most \( d \) edges. The time for computing \( \text{gain}(\cdot, \cdot) \) depends linearly on the length of the label of an edge. Note that \( G \) is a multigraph and that there exists an edge from \( v \) to \( w \) for every permutation of the characters of \( l((v, w)) \). As the maximal length of an edge label is \( p \), which is bounded by \( O(M/\mu) \), where \( \mu \) is the smallest mass of a character in \( \Sigma \), the time complexity for considering an outgoing edge (lines 8 and 10) is in \( O(p) \). Thus, the runtime of \( \text{DeNovo}_\triangle \) is in \( O(|V| \cdot |E| \cdot d \cdot p) \). \( \Box \)
When considering practical applications, the parameter $p$ depends on the data quality rather than on the size of the input $X$ and $M$. If we assume $p$ to be a constant, there are only $O(1)$ edges between two vertices and every vertex has only a constant out-degree. Hence, our algorithm matches the time complexity of Chen’s algorithm [8] unless the length of the edge labels grows asymptotically with the size of the input.

Computing the $k$ Best Solutions

We can find the $k$ best solutions for DeNovoSymDiff with a similar technique as proposed in [57]. We model the table $T$ as a directed acyclic graph that we call matrix graph. The edges in this graph correspond to all possible extension steps considered by DeNovo$\triangle$. The weight of an edge is equal to the additional score of the corresponding extension. A solution for the de novo sequencing problem corresponds to a path in this graph starting at the vertex representing the entry $T[0,(0,0)]$. The score of the solution is equal to the weight of the path.

The matrix graph $MG$ is a directed acyclic graph on vertices $V(MG) \subseteq (V \times E)$. For every entry $T[v,(a,b)]$ with $a \leq v \leq b$ and $v + b \leq M$, there is a vertex $v_{v,(a,b)}$ in $MG$. Every vertex $v_{v,(a,b)}$ has an outgoing edge to a vertex $v_{w,(a,b)}$ if $(v, w) \in E$ and $w \leq b$, respectively to a vertex $v_{b,(v,w')}$ if $(v, w') \in E$ and $w' > b$. Note that these edges correspond up to renaming to the extension steps in lines 8 and 10 of the pseudocode of DeNovo$\triangle$ (Algorithm 1). The vertex $v_{v,(a,b)}$ represents paths ending in $v$ and $(a,b)$. Outgoing edges represent all extensions with edges $(v, w)$ and $w \leq b$, respectively with edges $(v, w')$ with $w' > b$. The weight of each of these
edges is gain((v, w), (a, b)), respectively gain((v, w'), (a, b)), i.e. the additional score of adding the corresponding edge to the path in G ending in v.

A vertex \( v_{v,(a,b)} \) in MG is a terminal vertex if \( v = M - b \). A terminal vertex represents two paths that cannot be extended anymore, as they represent a prefix and a reversed suffix with a combined mass equal to \( M \). A path from \( v_{0,(0,0)} \) to a terminal vertex represents two substrings that can be concatenated to a string S of mass \( M \). The sum of the edge weights along this path is equal to score(S, X). Therefore, a solution for the de novo sequencing problem corresponds to a longest path from \( v_{0,(0,0)} \) to some terminal vertex in MG.

Similarly, the \( k \)-th best solution for the de novo sequencing problem corresponds to the \( k \)-th longest path from \( v_{0,(0,0)} \) to a terminal vertex in MG. We can apply Eppstein’s algorithm [28] to compute the \( k \) longest paths. Eppstein’s algorithm [28] computes the \( k \) shortest paths connecting a pair of vertices \( s \) and \( t \) in a directed acyclic graph with \( n \) vertices and \( m \) edges in \( O(n + m + k) \) time. The algorithm outputs an implicit representation of the paths and the sequence of edges of a path can be listed in time proportional to the length of the path. In order to compute the longest instead of the shortest paths, we multiply all edge weights with \(-1\). As the matrix graph can have multiple terminal vertices, but Eppstein’s algorithm only computes paths between two given vertices, we add a dummy vertex to the graph and connect all terminal vertices to this dummy vertex by directed edges with weight \( 0 \). Then, we compute the \( k \) longest paths between \( v_{0,(0,0)} \) and the dummy vertex in MG.

We can build \( MG \) while executing DeNovo\( Delta \) in the same time complexity \( O(|V| \cdot |E| \cdot d \cdot p) \), where \( V \) is the set of vertices and \( E \) the multiset of edges of \( G \), \( d \) is the maximal out-degree of a vertex.
in $G$ and $p$ is the maximal length of an edge label in $G$. The matrix graph has $O(|V| \cdot |E|)$ vertices and $O(|V| \cdot |E| \cdot d)$ edges. Hence, we can find the $k$ best solutions for the de novo peptide sequencing problem in $O(|V| \cdot |E| \cdot d \cdot p + k)$ time.
3.2.3 Algorithm for Multiple Ion Types

In the previous section, we studied the de novo sequencing problem in a simplified version. We assumed that a mass in $X$ corresponds exactly to the mass of the amino acid sequence of the measured fragment (Chapter 2). In real experiments, a mass in $X$ can have a small offset from the mass of the corresponding string. This is due to the fact that a peptide can split at different chemical bonds between two amino acids and can lose small neutral molecules (e.g. water, ammonia). In this section, we study a more general version of the de novo sequencing problem that considers such mass offsets with bounded maximal pairwise difference.

First, we formulate a more general variant of DeNovoSymDiff for a given set of possible mass offsets. We define the extended theoretical spectrum of a string $S$ as the set of all fragment masses with all possible mass offsets. As the possible offsets for prefixes and suffixes can differ, the extended theoretical spectrum of a string $S$ is not equal to the extended theoretical spectrum of the reversed string of $S$ and it is necessary to distinguish the prefix and the suffix in a path pair.

An important aspect of considering mass offsets is that they can alter the order of masses in $X$ with respect to the masses of the corresponding strings. This complicates the computation of the newly explained masses of an extension step. For example, the algorithm of Chen et al. [8] cannot deal with mass offsets that alter the order of the masses with respect to the masses of the corresponding strings. The order of the masses in $X$ does not change if the maximal difference of any two offsets is smaller than the smallest mass $\mu$ of a character in $\Sigma$. Our algorithm handles offsets with a maximal difference smaller than $2 \cdot \mu$. 

45
We model the extended theoretical spectrum as follows. Let $O_p$ and $O_s$ be the sets of all possible mass offsets $\delta \in \mathbb{Z}$ for a prefix fragment, respectively a suffix fragment. A prefix of a string $S$ with mass $m$ explains all masses in

$$OM(m, M) = \bigcup_{\delta \in O_p} (m + \delta) \cup \bigcup_{\delta' \in O_s} (M - m + \delta'),$$

(3.9)

where $M$ is the mass of $S$. The extended theoretical spectrum of a string $S$ is the set of all prefix and suffix masses with all possible offsets

$$TS_x(S) = \bigcup_{T \in \text{Pre}(S)} OM(m(T), m(S)).$$

(3.10)

We denote the maximal mass offset difference of two sets $(O_p, O_s)$ by $\gamma = \max_{\delta \in (O_p \cup O_s)} (\delta) - \min_{\delta' \in (O_p \cup O_s)} (\delta')$ and call an $(O_p, O_s)$ $\alpha$-basic set if $\gamma < \alpha \cdot \mu$.

Problem 2 (ExtendedDeNovoSymDiff). Let $\Sigma$ be an alphabet of characters with a mass $m(a) \in \mathbb{N}$ for each $a \in \Sigma$. Given a set $X = \{x_i \in \mathbb{N} \mid i = 1, \ldots, k\}$ of fragment masses, a peptide mass $M \in \mathbb{N}$, and $2$-basic sets $(O_p, O_s)$ of mass offsets, find a string $S$ of characters in $\Sigma$ with $m(S) = M$ that maximizes score$(S, X) = \sum_{m \in TS_x(S)} f_{\Delta}(m, X)$.

We can solve ExtendedDeNovoSymDiff by considering a multigraph $G_x = (V_x, E_x)$. In contrast to the multigraph $G$ defined in the previous section, $G_x$ contains up to $|O_p| + |O_s|$ vertices for each mass in $X$. For every $m \in X$, we consider every offset $\delta \in (O_p \cup O_s)$, assume that $m$ is the mass of a fragment with offset $\delta$ and add a vertex with the corresponding prefix mass to the graph. The
Figure 3.5: Extension of a prefix path $P$ by an edge $(v, w)$. Masses that are in $X$ are illustrated by vertical bars on the real line and other masses by crosses. For every vertex $m$ traversed by $P$, dotted lines point towards masses in $\text{OM}(m, M)$ on the line (masses that are larger than $M/2$ are omitted, mass offsets are 2-basic).

A multiset of edges is defined in the same way as for the multigraph $G$. A path in $G_x$ from vertex 0 to vertex $M$ corresponds to a string of mass $M$.

Our algorithm $\text{DeNovo}_{\Delta_x}$ solves $\text{ExtendedDeNovoSymDiff}$ by extending two paths that represent a prefix and a reversed suffix in the same way as $\text{DeNovo}_{\Delta}$: In every step, the algorithm extends the path representing the string with smaller mass. The extension of a path by an edge $(v, w)$ explains the masses

$$\text{TSe}_x((v, w), M) =$$

$$\begin{cases} 
\{ \text{OM}(v + m(T), M) \mid T \in \text{Pre}(l(v, w)), m(T) \neq 0 \} & \text{if } (v, w) \text{ is added to the prefix path}, \\
\{ \text{OM}(M - (v + m(T)), M) \mid T \in \text{Pre}(l(v, w)), m(T) \neq 0 \} & \text{otherwise}.
\end{cases}$$
Note that we distinguish prefixes and suffixes, as the extended theoretical spectrum is not necessarily symmetric. For an illustration, consider Figure 3.5. A prefix path \( P \) is extended by an edge \((v, w)\) labeled with a single character \( G \). The set \( TSe_x((v, w), M) \) contains only the masses in \( OM(w, M) \). Note that \( OM(w, M) \) contains no masses from \( OM(M - x, M) \) for any vertex \( x \) traversed before \( a \) by the suffix path \( Q \). In general, there are more masses in \( TSe_x((v, w), M) \) if the label of \((v, w)\) has multiple characters.

The set of newly explained masses by extending a path by an edge \((v, w)\) given the last edge \((a, b)\) of the second path is

\[
\text{New}((v, w), (a, b)) =
\begin{cases}
TSe_x((v, w), M) \setminus \left( OM(v, M) \cup \left( OM(M - a, M) \cup TSe_x((a, b), M) \right) \right) & \text{if } (v, w) \text{ is added to the prefix path,} \\
TSe_x((v, w), M) \setminus \left( OM(M - v, M) \cup OM(a, M) \cup TSe_x((a, b), M) \right) & \text{otherwise.}
\end{cases}
\]

Note that it is necessary to remove the masses in \( OM(v, M) \), respectively \( OM(M - v, M) \) from \( TSe_x((v, w), M) \). This is necessary regardless the fact that the masses explained by the substring with mass \( v \) are not considered in \( TSe_x((v, w), M) \), as \( OM(v, M) \cap OM(w, M) \) is not necessarily empty for 2-basic sets of mass offsets.

An example for the set of newly explained masses is shown in Figure 3.6. The path \( P \), which represents a prefix, is extended by an edge \((v, w)\). Let \( m \) be the mass of the first character of \( l(v, w) \). The masses explained by \( v + m \) in \( TSe_x((v, w), M) \) might also be explained by \( v \). However, \( m \) cannot explain any masses that are explained by
Figure 3.6: Extension of a path $P = (0, \ldots, v)$ that represents a prefix by an edge $(v, w)$. The mass of the first character of the label of $(v, w)$ is $m$. Any mass $m'$ traversed by $Q = (0, \ldots, a, b)$ before the source vertex $a$ of the last edge $(a, b)$ is at least $2\mu$ smaller than $w + m$.

some mass $m'$ traversed by the other path before the source vertex $a$ of the last edge $(a, b)$, as the mass difference of $v + m$ and $m'$ is at least $2\mu$.

DeNovo$_\Delta x$ computes an optimal path in $G_x$ in the same fashion as DeNovo$_\Delta$ described above. The algorithm solves ExtendedDeNovoSymDiff in time $O(|V_x| \cdot |E_x| \cdot d \cdot p \cdot |O|)$, where $d$ is the maximal out-degree of a vertex in $G_x$, $p$ is the maximal length of an edge label, and $|O| = |(O_p \cup O_s)|$ is the number of possible mass offsets.

3.2.4 Remark about Scoring Functions

A scoring function for the de novo sequencing problem compares the theoretical spectrum of a string $S$ with the experimental spectrum measured by the mass spectrometer. In the previous sections, we considered very intuitive scoring functions that count the number of masses in $\text{TS}(S) \cap X$ (shared peaks count, $f_{scp}$), respectively in $\text{TS}(S) \Delta X$ (symmetric difference, $f_\Delta$). These scoring functions do
not consider any other information about the measured masses, such as the signal intensity, the type of the fragment, etc.

There exist several, more evolved scoring functions [61, 12] that consider, for example, the signal intensity \( I(m) \in \mathbb{R}^+ \) of each mass \( m \in X \) measured by the mass spectrometer. Instead of only counting the number of explained masses that are measured in the experiment, the signal intensities of these masses are summed up. That is, a weighted shared peaks count with

\[
f_{\text{wscp}}(m, X) = \begin{cases} 
I(m) & \text{if } m \in X, \\
0 & \text{if } m \notin X
\end{cases}
\]

is maximized. The intuition for this scoring function is that one prefers to explain fragment mass with high intensities, as the intensity corresponds to the abundance of the fragment and as low-intensity signals are more likely to originate from contaminants or measurement noise.

A weighted variant of the symmetric difference scoring function can be defined analogously. However, as we do not only consider measured masses, we define a constant penalty intensity \( p \in \mathbb{R} \) for all masses \( m \notin X \).

\[
f_{\text{w\w}}(m, X) = \begin{cases} 
I(m) & \text{if } m \in X, \\
p & \text{if } m \notin X
\end{cases}
\]

In practice, one would rather use a non-constant penalty \( p(m) \) with some underlying model for predicting the signal intensity of a mass \( m \) based on the mass and the type of the fragment, the amino acids adjacent to the cleavage sites, and other factors. To incorporate the
weighted variant (Equation (3.12)) in the algorithm, it is sufficient to replace $f_\Delta()$ by the weighted variant $f_{w_\Delta}()$ in Equation (3.6).

### 3.2.5 Experimental Evaluation

We implemented DeNovo\(\Delta\) and studied the quality of its solution when using the shared peaks count scoring function and the symmetric difference scoring function. We chose DeNovo\(\Delta\) rather than DeNovo\(\Delta_x\) in our experiments to clearly expose the effect of the symmetric difference scoring function. We do not aim for an advanced software toolkit for de novo sequencing, but rather propose a new fundamental scoring model that does not come at a substantial extra computational cost. The implementation is available under a BSD license at github.com/ttschager.

**Dataset** We considered spectra of 422 synthetic peptides that have been acquired in data-dependent acquisition (DDA) mode. The data is part of the SWATH-MS Gold Standard (SGS) dataset (PASS00289 at peptideatlas.org,[80]). First, we analyzed the spectra using the database search tool Comet[26] and a database containing only the sequences of the 422 synthetic peptides. The Comet search results were further validated using peptideprophet, which provides a statistical estimation for the false discovery rate [47]. We considered a peptide to be identified if the identification probability as returned by peptideprophet was higher than 90%.

For our evaluation, we considered all spectra, where Comet was able to identify the expected synthetic peptide sequence. We did not consider spectra, where Comet reported a sequence with amino acid modifications or a sequence that was not ending with amino acid R or K, as the
current implementation of our algorithm is not able to reconstruct such sequences. If Comet identified a peptide in multiple spectra, we considered all of them for our comparison, as it is not clear how to choose one of these spectra as the representative for the peptide. In total, we considered 944 spectra for our evaluation. We considered the raw profile data and implemented the merging algorithm proposed in [15] to reduce the size of the graph (i.e. centroiding). A peptide is considered to be identified by our algorithm if the algorithm reports the same sequence as Comet as the best-scoring sequence.

Runtime While we are not primarily interested in runtime differences of the shared peak count and the symmetric difference scoring function, we observed that both algorithms have very similar running times (on an Intel Core i5-3317U with 4 GB RAM, Figure 3.7). Note that the running times of state-of-the-art software packages as PepNovo [30], PEAKS [60], and especially Novor [58] are by magnitudes faster. The comparison indicates that considering the symmetric difference instead of the shared peak count does not come at a substantial extra computational cost.

Identification rates Comet was able to identify 354 of the 422 synthetic peptides in the considered spectra. Considering the shared peaks count (SPC), our algorithm identified 227 peptides, whereas it was able to identify 270 peptides considering the symmetric difference (SymDiff) scoring model (Figure 3.8).

For a more detailed comparison, we first considered the position of the true sequence in the list of candidate solutions (sorted by their scores) and secondly the similarity of the best-scoring sequence with the true sequence. For the first comparison, our algorithm computed all solutions with a score of at least 90% of the maximum score. For
Figure 3.7: Comparison of the running times of DeNovoΔ for analyzing a single spectrum using the shared peak count (SCP), respectively the symmetric difference (SymDiff) scoring model. All running times were measured considering the preprocessed (centroided, peak-picked) spectra of the DDA SGS dataset.

In the second comparison, we measured the similarity of two sequences by considering their sets of prefix masses. The recall of a reported sequence is the number of prefix masses it has in common with the true sequence divided by the number of prefix masses of the true sequence:
Figure 3.8: Number of peptides that were identified by DeNovo\(\Delta\) when (i) maximizing the shared peaks count (SCP) and (ii) minimizing the symmetric difference (SymDiff).

\[
\text{recall} = \frac{\text{number of correct prefix masses}}{\text{number of prefix masses of the true sequence}}. \quad (3.13)
\]

Figure 3.9 shows the position of the true sequence (as annotated by Comet) in the list of candidate sequences (sorted by their score). The complete true sequence was among the top 10 sequences in 49.8% of the spectra considering the shared peaks count and in 67.1% of the spectra considering the symmetric difference.

Figure 3.10 depicts the similarity of the best-scoring sequence compared to the sequence identified by Comet. If there were multiple best-scoring sequences, we considered the one with the highest recall. In 60.4% of the considered spectra, our algorithm reported a sequence with a recall of at least 80% when considering the shared peaks count. Considering the symmetric difference, the best-scoring sequence had a recall of at least 80% in 72.9% of the considered spectra.
Discussion

Although it has already been pointed out [15] that penalizing the fact that an explained mass is not measured improves the performance of algorithms for peptide identification, minimizing the symmetric difference of the set of explained masses and the set of measured masses has not been studied before. We did not focus on developing a sophisticated scoring function for this scoring model, but rather
<table>
<thead>
<tr>
<th>Number of spectra</th>
<th>= 100%</th>
<th>&gt;= 95%</th>
<th>&gt;= 80%</th>
<th>&gt;= 75%</th>
<th>&gt;= 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPC</td>
<td>447</td>
<td>520</td>
<td>570</td>
<td>616</td>
<td>744</td>
</tr>
<tr>
<td>SymDiff</td>
<td>47.4%</td>
<td>55.1%</td>
<td>60.4%</td>
<td>65.3%</td>
<td>78.8%</td>
</tr>
</tbody>
</table>

Figure 3.10: Recall of the best-scoring sequence reported by \texttt{DeNovoΔ} when considering the shared peaks count (SCP) and the symmetric difference (SymDiff).

focused on very simple scoring functions and showed that without substantial extra computational effort, moving from shared peaks count to symmetric difference as optimization goal can improve the identification rates for de novo peptide sequencing.

Instead of only considering if a mass has been measured by the instrument or not, one can also account for the measured signal in-
tensities. In the intensity-based variants of the considered scoring functions, we do not only look at the size of the sets $TS(S) \cap X$ and $TS(S) \setminus X$, but we chose to sum up the intensities of the corresponding signals. The intensity-based variant of the shared peaks count is equivalent to the score proposed in [61]. However, the corresponding software PEAKS [60] uses additional features for scoring and is therefore not suitable for a comparison with our current implementation.

For this variant, one has to introduce a parameter $p(m)$ for penalizing an explained mass $m$ that is not in $X$ when considering the symmetric difference. Setting $p(m) = 0$ for all $m$ is equivalent to considering the intensity-based weighted shared peaks count. Rather than penalizing equally all explained masses that are not measured, one can incorporate some model for predicting the signal intensities [25, 18, 17]. Similarly, in order to consider losses of neutral molecules or other types of fragments with mass offsets, one would need to define an appropriate penalty if the corresponding mass in the extended theoretical spectrum is not measured. For example, this penalty should depend on the type of the fragment and on whether a neutral loss is involved. Our algorithm can incorporate such aspects and gives us the possibility to develop more sophisticated scoring functions that model the fragmentation process more accurately. In an experiment with a constant penalty parameter, the performance of our algorithm improved using the intensity-based variants. However, our algorithm was not able to identify more peptides.

In our evaluation, we used the raw profile data measured by the instrument and a simple merging algorithm [15] to reduce the size of $X$. We additionally tested our algorithm on data that has been preprocessed (centroided) by qtofpeakpicker [82]. Our algorithm was able
to identify 237 peptides with the shared peaks count scoring function and 284 peptides with the symmetric difference scoring function in these preprocessed spectra. The identification rates declined considering the intensity-based variants of the scoring functions. We refer to [90] for more details about these experiments. We suppose that a more evolved model for penalizing explained masses that are not measured would be necessary to further improve the identification rates.
3.3 De Novo Sequencing with Liquid Chromatography

The information obtained from the chromatographic separation in the first step of the LC-MS/MS experiment is usually not considered by de novo peptide sequencing algorithms. However, this information has proven to be useful for peptide identification. For example, several studies propose to use it in a filtering step after a database search to increase the confidence of identification and to identify false positive identifications \[74, 87, 73, 78, 77\].

Predicting the retention time of a peptide, i.e. the time the peptide needs to traverse the liquid chromatography column, from its amino acid sequence is a challenging task \[4, 84, 67\]. The retention time contains information about parts of a sequence that cannot be resolved by mass spectrometry (e.g. amino acids and fragments with equal masses, but different retention times). Moreover, it is available without additional experimental effort. However, to the best of our knowledge, the retention time information has not been considered by de novo peptide sequencing algorithms.

We formulate and study a de novo sequencing problem that integrates the retention time as an additional constraint and does not require filtering many candidates. We are interested in a sequence that both matches the experimental spectrum and the measured retention time and focus on three additive retention time prediction models. Considering the retention time information comes at the cost of higher computational effort and requires additional parameters for retention time prediction (either estimated from suitable datasets or taken from the literature). Yet, our experimental evalua-
(a) Linear model
\[ t_{\text{lin}}(S) = t(A) + t(I) + t(A) + t(G) + t(A) + t(K) \]

(b) Position-dependent model (γ = 2)
\[ t_{\text{pos}}(S) = t_{\text{pre}}(A, 1) + t_{\text{pre}}(I, 2) + t(A) + t(G) + t_{\text{suf}}(A, 2) + t_{\text{suf}}(K, 1) \]

(c) Neighborhood-based model
\[ t_{\text{nei}}(S) = t(−, A) + t(A, I) + t(I, A) + t(A, G) + t(G, A) + t(A, K) + t(K, −) \]

Figure 3.11: Retention time prediction for the string \( S = \text{AIAGAK} \). (a) In the linear model, the retention time of a string is the sum of its character’s coefficients. (b) In the position-dependent model (with \( \gamma = 2 \)), the position of the first and the last two characters is also considered. (c) The neighborhood-based model uses coefficients for all pairs of consecutive characters. The first and the last character have additional coefficients, as they have only one adjacent character.

3.3.1 Problem Definition

Retention time prediction models We define three simple models for predicting the retention time of a string \( S = a_1 \ldots a_n \) (Figure 3.11). The first model is a simple additive model with one coefficient for each character. The model only considers the character frequencies of a string and has been proposed in [38]. It served as starting point for the development of more evolved prediction models [53, 52].
The other two models consider additional factors that affect the retention time of a peptide. The position of the characters in the string is especially important for the first and the last few positions in the string [53, 52]. The second model considers distinct coefficients for the characters at the beginning and the end of the string.

The nearest neighborhood of a character can also affect its retention time coefficient [52, 85]. The third model considers the influence of a character’s direct neighborhood by considering coefficients for pairs of consecutive characters instead of coefficients for individual characters.

**Linear model** Every character \( a \in \Sigma \) has a retention time coefficient \( t(a) \in \mathbb{Z} \). The retention time of a string \( S \) is the sum of the retention time coefficients of its characters,

\[
t_{\text{lin}}(S) := \sum_{i=1}^{n} t(a_i). \tag{3.14}
\]

**Position-dependent model** Characters at the first \( \gamma \) and the last \( \gamma \) positions of a string, where \( 1 \leq \gamma \leq \lfloor \frac{n}{2} \rfloor \), have distinct retention time coefficients. For \( i \leq \gamma \), we denote the retention time coefficient of the \( i \)-th character by \( t_{\text{pre}}(a_i, i) \in \mathbb{Z} \) and the coefficient of the \((n - i + 1)\)-th character by \( t_{\text{suf}}(a_{n-i+1}, i) \in \mathbb{Z} \). The retention time of a string \( S \) is the sum of the corresponding retention time coefficients,

\[
t_{\text{pos}}(S) := \sum_{i=1}^{\gamma} t_{\text{pre}}(a_i, i) + \sum_{j=\gamma+1}^{n-\gamma} t(a_j) + \sum_{k=1}^{\gamma} t_{\text{suf}}(a_{n-k+1}, k). \tag{3.15}
\]
**Neighborhood-based model** The model uses retention time coefficients \( t(a, b) \in \mathbb{Z} \) for pairs of characters \( a, b \in \Sigma \) that are consecutive in a given string \( S \). The first and the last character \( a_1 \) and \( a_n \) of \( S \) have additional coefficients \( t(-, a_1), t(a_n, -) \in \mathbb{Z} \), as these characters have only one adjacent character in \( S \). The retention time of \( S \) is the sum of all these coefficients,

\[
t_{\text{nei}}(S) := t(-, a_1) + \left( \sum_{i=1}^{n-1} t(a_i, a_{i+1}) \right) + t(a_n, -). \tag{3.16}
\]

The retention time coefficients for all three models can either be estimated from experimental data or taken from the literature. We use a simple method for estimating the coefficients in the experimental evaluation and discuss limiting aspects of this method below.

**Problem definition** **DeNovoSymDiff** (Section 3.2.1) is the problem of finding a string \( S \) with a given mass \( M \) that minimizes for a given fragmentation pattern \( X \) the size of \( \text{TS}(S) \triangle X \). We extend this problem and additionally consider the measured retention time \( T \) and a retention time prediction function \( t_\ast : \Sigma^* \rightarrow \mathbb{Z} \). A function \( t_\ast() \) can return negative values, as a substring can have a negative effect on the retention time of a string.

**Problem 3 (DeNovoRT).** Let \( \Sigma \) be an alphabet of characters, with a mass \( m(a) \in \mathbb{N} \) for each \( a \in \Sigma \). Given a peptide mass \( M \in \mathbb{N} \), a retention time \( T \in \mathbb{N} \), a tolerance parameter \( \varepsilon \geq 0 \), and a fragmentation pattern \( X = \{ x_i \in \mathbb{N} \mid i = 1, \ldots, k \} \), find a string \( S \) of characters in \( \Sigma \) with \( m(S) = M \) and \( |t_\ast(S) - T| \leq \varepsilon \) that minimizes \( |	ext{TS}(S) \triangle X| \) among all strings with mass \( M \) and a retention time \( t_\ast(S) \in [T - \varepsilon, T + \varepsilon] \).
3.3.2 Algorithms

In the following subsections, we develop algorithms for the problem DeNovoRT and the three retention time prediction models discussed in the previous section. The algorithms are based on the algorithm DeNovo△ for minimizing the symmetric difference between expected and measured fragment masses discussed in Section 3.2. This algorithm is a graph-based approach that constructs a multigraph $G$ based on the fragmentation pattern $X$. Then, it iteratively extends a path pair (Definition 2), i.e. two paths in this graph, one representing a prefix and the other representing a reversed suffix of the solution. The algorithm always extends the path representing the string of smaller mass (balanced extension, Definition 1). A path pair representing an optimal string for the problem DeNovoSMDiff has optimal substructures and can be efficiently computed by dynamic programming.

To take into account the retention time information, we have to consider three aspects: First, we need to define the predicted retention time of a path pair in $G$. Second, we have to compute the effect of a balanced extension on the predicted retention time of a path pair. Third, we need to find optimal substructures of paths from 0 to $M$ in $G$ with an optimal score and a feasible predicted retention time.

Algorithm for the Linear Prediction Model

In this section, we consider the linear retention time prediction model. Let $(P, Q)$ with $P = (0, \ldots, v)$ and $Q = (0, \ldots, a, b)$ be a path pair in the multigraph $G$. From $v \leq b$ follows $a \leq v$ (Lemma 1) and we know that $v + b \leq M$ by the definition of path pairs and balanced
extensions (Definitions 1 and 2). We define the retention time $t$ of a path pair $(P, Q)$ as the sum of the retention times of the substrings that $P$ and $Q$ represent, i.e. $t = t_{\text{lin}}(l(P)) + t_{\text{lin}}(l(Q))$. The retention time $t'$ of a path pair obtained from $(P, Q)$ by applying a balanced extension by some edge $(v, w)$ can be computed by adding the retention time of the edge label $l(v, w)$ to $t$, i.e. $t' = t + t_{\text{lin}}(l(v, w))$. That is, we only need $t$ and the edge label $l(v, w)$ for computing $t'$.

However, it is not sufficient to only store the optimal score of any path pair ending in $v$, respectively $(a, b)$, and the retention time of such a path pair to compute a solution for DeNovoRT. There can be multiple path pairs ending in the same vertex and the same edge with different retention times. Consider an optimal string and its sequence of path pairs computed by the algorithm. A path pair $P = (0, \ldots, v)$ and $Q = (0, \ldots, a, b)$ in this sequence does not necessarily have an optimal score among all path pairs ending in $v$ and $(a, b)$. Nevertheless, its score is optimal among all path pairs with the same retention time that end in $v$ and $(a, b)$. Therefore, we need to store for each possible retention time $t$ the optimal score of a path pair ending in the vertex $v$ and the edge $(a, b)$.

The algorithm $\text{DeNovo} \triangle \text{Lin}$ (Algorithm 2) for DeNovoRT with the linear prediction model computes a dynamic programming table $T$ of size $|V| \cdot |E| \cdot |RT_M|$, where $|RT_M|$ denotes the number of possible retention times for a string of mass $M$. An entry $T[v, (a, b)][t]$ contains the optimal score for a path pair ending in $v$, respectively $(a, b)$, with retention time $t$.

For a given vertex $v$ and an edge $(a, b)$, the algorithm performs balanced extensions by all outgoing edges $(v, w)$ of $v$. For every balanced extension and every feasible retention time $t$, the algorithm then computes the new retention time $t'$ and the new score of the
Algorithm 2: DeNovoΔLin

foreach \((a, b) \in E \text{ and } v \in V\) do  
\[T[v, (a, b)] \leftarrow \text{array with entries } -\infty \text{ for each feasible retention time } t\]  
\[T[0, (0, 0)][0] \leftarrow 2\]

foreach \(v \in V \text{ in ascending order}\) do  
foreach \((a, b) \in E \text{ in lexicographically ascending order}\) do  
foreach entry \(t\) in \(T[v, (a, b)]\) do  
\[t' \leftarrow t + t_{\text{lin}}(l(v, w))\]
\[\text{new} \leftarrow T[v, (a, b)][t] + \text{gain}((v, w), (a, b))\]

if \(w \leq b\) then  
\[T[w, (a, b)][t'] \leftarrow \max(T[w, (a, b)][t'], \text{new})\]
else
\[T[b, (v, w)][t'] \leftarrow \max(T[b, (v, w)][t'], \text{new})\]

resulting path pair and updates the corresponding entry in the table. We can see by an inductive argument that the optimal scores in the table are computed correctly. As the base case, we note that \(T[0, (0, 0)][0] = 2\) is correct, as an empty path pair explains the masses \(\{0, M\} \subseteq X\) and has retention time 0. As soon as the entry \(T[v, (a, b)]\) is reached in line 6, all optimal scores for path pairs ending in vertex \(v\) and edge \((a, b)\) have been computed. This holds by induction, as every possible balanced extension leading to a path pair ending in \(v\) and \((a, b)\) has already been considered (given the optimal
score of a preceding path pair). Moreover, the array in \( T[v, (a, b)] \) is not further modified as soon as the algorithm reaches the vertex \( v \) and the edge \((a, b)\) in line 6. Therefore, the invariant holds that, if the algorithm considers a vertex \( v \) and an edge \((a, b)\) in line 6, the corresponding entry \( T[v, (a, b)] \) contains the optimal score for each feasible retention time.

After computing all entries \( T[v, (a, b)] \), we can find the score of an optimal solution by iterating over all entries \( T[M - b, (a, b)][t] \) for \((a, b) \in E\) and all feasible retention times \( t \in [T - \varepsilon, T + \varepsilon] \). We can reconstruct a corresponding optimal string starting from this entry.

**Time complexity** Compared to \( \text{DeNovo\textDelta} \), the additional overhead of \( \text{DeNovo\textDeltaLin} \) (lines 7 and 8 in Algorithm 2) is to iterate over all feasible retention times \( t \) for each entry \( T[v, (a, b)] \) and compute the new retention time \( t' \). The number of scores to be stored varies depending on the entry and the retention time coefficients of the prediction model. For a path pair ending in \( v \), respectively \((a, b)\), we have to consider all retention times in \([rt_{\text{min}} \cdot (v + b), rt_{\text{max}} \cdot (v + b)]\), where \( rt_{\text{min}} \) (\( rt_{\text{max}} \)) is the minimum (maximum) retention time per mass unit. For example, we only store one optimal score in entry \( T[0, (0, 0)] \), but up to \([rt_{\text{max}} \cdot M - rt_{\text{min}} \cdot M] \) scores in entries \( T[M - b, (a, b)] \) for \((a, b) \in E\).

The time complexity of \( \text{DeNovo\textDelta} \) is in \( O(|V| \cdot |E| \cdot d \cdot p) \), where \( d \) is the maximal out-degree of a vertex in \( G \) and \( p \) is the maximal length of an edge label (Theorem 2). The time complexity of \( \text{DeNovo\textDeltaLin} \) has an additional multiplicative term \(|RT_M|\) that denotes the number of possible retention times for a string of mass \( M \). In practice, most entries \( T[v, (a, b)] \) contain only few scores, as we only store the score
for a retention time $t$ if there is a path pair ending in $v$ and $(a, b)$ with predicted retention time $t$. It is advisable to use a memory-efficient data structure to reduce the memory consumption of the algorithm.

**Algorithm for the Position-dependent Model**

In the position-dependent prediction model, the retention time of a string $S$ is not equal to the retention time of all permutations of $S$. This is due to the fact that the retention time coefficient of a character in one of the first and one of the last $\gamma$ positions of the string may be different from the coefficient of the same character at another position. Therefore, we have to distinguish the prefix and the suffix path of a path pair $(P, Q)$, with $P = (0, \ldots, v)$, $Q = (0, \ldots, a, b)$, and $a \leq v \leq b$, in order to compute its predicted retention time. This was not necessary for DeNovoLin, as the score and the predicted retention time (in the linear prediction model) does not depend on which of both paths represents the prefix.

Let us assume that $P$ is the prefix path and $Q$ is the suffix path of a path pair $(P, Q)$. We compute the retention time of $(P, Q)$ by summing the retention times $t_P$ and $t_Q$ of the path labels,

$$
t_P := \sum_{a_i \in l(P)} \begin{cases} t_{\text{pre}}(a_i, i) & i \leq \gamma \\ t(a_i) & i > \gamma \end{cases}
$$

$$
t_Q := \sum_{a_j \in l(Q)} \begin{cases} t_{\text{ suf}}(a_j, j) & j \leq \gamma \\ t(a_j) & j > \gamma \end{cases} \tag{3.17}
$$

If we want to update the retention time after a balanced extension of $(P, Q)$ by an edge $(v, w)$, we have to compute the retention time of the edge label $l(v, w)$. This retention time depends on whether the edge label contains some of the first or the last $\gamma$ characters of a
solution string $S$ of mass $M$. However, there can be multiple such solution strings resulting from different further balanced extensions of this path pair.

We can decide whether $l(v, w)$ contains some of the first $\gamma$ characters given the length $k$ of $l(P)$ without knowing the solution string $S$. If $k \geq \gamma$, the edge label clearly does not contain any of the first $\gamma$ characters of any solution resulting from extending $(P, Q)$. Likewise, we know that $l(v, w)$ contains none of the $\gamma$ last characters if $l(Q)$ has more than $\gamma$ characters. However, if $l(Q)$ has less than $\gamma$ characters, we cannot decide whether $l(v, w)$ contains some of the last $\gamma$ characters without knowing the length of $S$.

Let us assume for now that $l(v, w)$ does not contain some of the last $\gamma$ characters of the solution. The retention time of the new path pair resulting from the balanced extension of $(P, Q)$ by the edge $(v, w)$ is

$$t' = t + \sum_{a_i \in l(v, w)} \begin{cases} t_{\text{pre}}(a_i, i) & i + k \leq \gamma \\ t(a_i) & i + k > \gamma \end{cases}$$

(3.18)

In the case, where $P$ is the suffix path, $t_{\text{pre}}(a_i, i)$ is replaced by $t_{\text{suf}}(a_i, i)$ in the above equation.

It is important that the above assumption holds for every balanced extension leading to a solution string $S$. Otherwise, the retention time of the new path pair is not computed correctly. We cannot check if our assumption holds while computing the new retention time after a balanced extension. However, given a solution string $S$ and a path pair that represents a prefix and a suffix of $S$, we can check if either the balanced extension leading to this path pair or a preceding balanced extension does not satisfy the assumption. If so,
either the prefix or the suffix path label has at least \( n - \gamma \) characters, where \( n \) is the length of \( S \). This also holds for all subsequent path pairs, as we only add characters to path labels.

When reconstructing a solution from the dynamic programming table, we have to additionally check, if one of the path labels has \( n - \gamma \) or more characters, before we combine them to a solution string. If so, the assumption was not fulfilled at some step and we discard this solution, as its retention time was not computed correctly. Note that we cannot consider these strings, unless they can be constructed by another sequence of balanced extensions. However, it is very unlikely that the assumption is not fulfilled in practice, as we consider small values of \( \gamma \). We never observed such a situation in our evaluation using \( \gamma = 2 \).

Given the sequence of path pairs of an optimal solution, a path pair in this sequence has an optimal score among all path pairs with the same retention time. However, we have to store some additional information to compute a solution with respect to the position-dependent prediction model. First, we have to store whether \( P \) is a prefix or a suffix path. Second, we have to store the length of both path labels, unless they are larger than \( \gamma \).

DeNovoΔPoS (Algorithm 3) stores the optimal scores of path pairs ending in \( v \) and \((a, b)\) in an array with an entry for every retention time \( t \), the lengths \( \alpha \) and \( \beta \) of the path labels and a Boolean variable \( \text{pre} \) indicating if the path ending in \( v \) is the prefix or the suffix path. We store the length of the path labels only up to length \( \gamma \), as the exact length is only important as long as the path labels have less than \( \gamma \) characters.

If the algorithm reaches an entry \( T[v, (a, b)] \) in line 7, all optimal scores for path pairs ending in vertex \( v \) and edge \((a, b)\) have been
Algorithm 3: DeNovo△Pos

foreach \((a, b) \in E\) and \(v \in V\) do

\[ T[v, (a, b)] \leftarrow (|RT_M| \times \gamma \times \gamma \times 2)\text{-array with entries } -\infty \]

\[ T[0, (0, 0)][0, 0, 0, 0] \leftarrow 2 \]

foreach \(v \in V\) in ascending order do

foreach \((a, b) \in E\) in lexicographically ascending order do

foreach entry \((t, \alpha, \beta, \text{pre})\) in \(T[v, (a, b)]\) do

\[ t' \leftarrow \text{retention time of resulting path pair} \]

if \(\text{pre}\) then

\[ \alpha' \leftarrow \max(\gamma, \alpha + |l(v, w)|) \]

\[ \beta' \leftarrow \beta \]

else

\[ \alpha' \leftarrow \alpha \]

\[ \beta' \leftarrow \max(\gamma, \beta + |l(v, w)|) \]

new \[= T[v, (a, b)][t, \alpha, \beta, \text{pre}] + \text{gain}((v, w), (a, b)) \]

if \(w \leq b\) then

\[ T[w, (a, b)][t', \alpha', \beta', \text{pre}] \leftarrow \max(T[w, (a, b)][t', \alpha', \beta', \text{pre}], \text{new}) \]

else

\[ T[b, (v, w)][t', \alpha', \beta', \neg\text{pre}] \leftarrow \max(T[b, (v, w)][t', \alpha', \beta', \neg\text{pre}], \text{new}) \]
computed correctly, as all balanced extensions leading to such path pairs have already been considered. Given the optimal score of a path pair, the algorithm performs every possible balanced extension with an outgoing edge of \( v \), computes the new score and the new retention time, and updates the corresponding entries. We reconstruct a solution starting from a path pair ending in some vertex \( M - b \) and some edge \( (a, b) \). The algorithm additionally verifies that both the prefix and the suffix path label have more than \( \gamma \) characters. \texttt{DeNovo\textsc{PoS}} considers at most \( 2 \cdot \gamma^2 \cdot |RT_M| \) optimal scores for each table entry \( T[v, (a, b)] \), where \( |RT_M| \) is the number of possible retention times for a string of mass \( M \). Therefore, the running time is in \( O(|V| \cdot |E| \cdot |RT_M| \cdot \gamma^2 \cdot d \cdot p) \), where \( d \) is the maximal out-degree of a vertex in \( G \) and \( p \) is the maximal length of an edge label.

Algorithm for the Neighborhood-based Model

The neighborhood-based model considers pairs of consecutive characters for predicting the retention time of a given string \( S \). We define the predicted retention time of a path pair \((P, Q)\), where \( P \) is the prefix path and \( Q \) is the suffix path, as follows. The retention time of a path label takes only into account one coefficient for the last character of the path label, as the other coefficient depends on the next balanced extension of this path or on the last character of the other path label. Note that we have to consider that one of the path labels represents the reversed suffix of the solution. The retention time of \((P, Q)\) with path labels \( l(P) = p_1, \ldots, p_n \) and \( l(Q) = q_1, \ldots, q_m \) is
The retention time $t$ of a path pair $(P, Q)$ is the sum of the retention time coefficients up to the last characters $p_2$ and $q_3$. A balanced extension with an edge labeled with label $l_1l_2$ leads to the path pair $(P', Q)$. If $m(l(P')) + m(l(Q)) = M$, a solution can be constructed by concatenating $l(P')$ and the reversed string of $l(Q)$.

The retention time after a balanced extension of $(P, Q)$ as follows. Consider a balanced extension of the prefix path $P$ by an edge $(v, w)$ with $l(v, w) = l_1 \ldots l_k$. Let $p_n$ be the last character of $l(P)$. The retention time $t'$ of the new path pair resulting from the balanced extension is

$$t' = t_{\text{nei}}(P, Q) + t(p_n, l_1) + \sum_{i=1}^{k-1} t(l_i, l_{i+1}). \quad (3.20)$$

The retention time after a balanced extension of the suffix path $Q$ is defined analogously (again considering that $l(Q)$ is a reversed suffix).
Note that the retention time of a solution $S$ is not the sum of the retention times of a prefix of $S$ and its complementary suffix. We have to additionally consider the coefficient of the last character of the prefix and the first character of the suffix, which are consecutive in $S$. If we combine the path labels of a path pair $(P', Q)$ to a string $S$ (Figure 3.12), the retention time of $S$ is $t_{nei}(P', Q) + t(p_n, q_m)$, where $p_n$ and $q_m$ are the last characters of $P$ and $Q$.

DeNovo$\triangle$Nei (Algorithm 4) stores for every path pair $(P, Q)$ ending in vertex $v$ and edge $(a, b)$ the optimal score for each retention time $t$, last character $p$ of the path ending in $v$, and a Boolean variable pre indicating if $P$ is the prefix path. As a base case, the algorithm stores the optimal score for a path pair ending in vertex 0 and the loop edge $(0, 0)$ as $T[0, (0, 0)][0, 0] = 2$. The algorithm considers the vertices and edges of $G$ in ascending order. After considering all possible path pairs, the optimal score can be computed by considering all entries $T[M - b, (a, b)]$ and the feasible solutions for path pairs ending in these vertices and edges.

The algorithm considers at most $2 \cdot |\Sigma| \cdot |RT_M|$ optimal scores for each pair of a vertex $v$ and an edge $(a, b)$, where $|RT_M|$ is the number of possible retention times for a string of mass $M$ and $|\Sigma|$ is the size of the considered alphabet. The running time of DeNovo$\triangle$Nei is in $O(|V| \cdot |E| \cdot |RT_M| \cdot |\Sigma| \cdot d \cdot p)$, where $d$ is the maximal out-degree of a vertex, $p$ is the maximal length of an edge label, and $|RT_M|$ is the number of feasible retention times for a string of mass $M$.

### 3.3.3 Experimental Evaluation

In this section, we study the performance of our algorithms. We focus on clearly exposing the effect of considering the retention time
Algorithm 4: DeNovoΔNei

```
foreach (a, b) ∈ E and v ∈ V do
    T[v, (a, b)] ← (|RTM| × |Σ| × 2)-array with entries −∞
T[0, (0, 0)][0, −, 0] ← 2

foreach v ∈ V in ascending order do
    foreach (a, b) ∈ E in lexicographically ascending order do
        foreach (v, w) ∈ E with w + b ≤ M do
            foreach entry (t, p, pre) in T[v, (a, b)] do
                t′ ← retention time of resulting path pair
                new ← T[v, (a, b)][t, p, pre] + gain((v, w), (a, b))
                if w ≤ b then
                    p′ ← last character of l(v, w)
                    T[w, (a, b)][t′, p′, pre] ← max(T[w, (a, b)][t′, p′, pre], new)
                else
                    p′ ← last character of l(a, b)
                    T[b, (v, w)][t′, p′, ¬pre] ← max(T[b, (v, w)][t′, p′, ¬pre], new)
```

74
information. We compare the identification rates of the proposed algorithms with the identification rates of DeNovo\(\Delta\) (Section 3.2). Note that we use a very simple scoring function that only considers if a mass has been measured by the instrument, but no other information, such as the intensity of the signal. While this is sufficient for studying the effect of considering the retention time information, such a scoring function is typically not suitable for real applications.

**Dataset**  We use mass spectra from the SWATH-MS Gold Standard (SGS) dataset (PASS00289 at peptideatlas.org, [80]). Specifically, we consider the 944 spectra of synthesized peptides acquired in data-dependent acquisition mode that we considered in Section 3.2.5 for evaluating the algorithm DeNovo\(\Delta\). The raw profile spectra were centroided (peak-picked) using the tool qtofpeak-picker [82]. The spectra have been analyzed using the database search tool Comet [26] using the very restricted database containing only the 422 synthesized peptides. We only considered spectra from doubly-charged peptides (as reported by Comet). Moreover, the algorithms considered all measured fragment masses to be singly charged. Considering fragments with higher charge states is possible, but requires additional data preprocessing to convert the measured mass-to-charge ratios to the masses. Peptideprophet [47] has been used to validated the results.

We used the sequences identified by Comet as gold standard and considered a peptide to be identified by one of the considered algorithms, if the exact sequence has been computed as the best-scoring solution, respectively one of the 5, 10, or 100 best-scoring solutions.

**Retention time coefficient estimation**  We are mainly interested in the algorithmic problem of using retention time information for
de novo sequencing and do not focus on efficient procedures for estimating the coefficients of retention time prediction models. We use linear regression for estimating the coefficients for our three retention time models.

We randomly split the 944 spectra into a training set with 80% of the spectra (755 spectra) and a test set with the remaining 20% of the spectra (189 spectra). We use the training set for estimating the retention time coefficients and the test set for selecting a tolerance parameter $\varepsilon$. In a linear regression approach, we choose the coefficients such that the sum of the squared loss $\sum_i (T_i - t(S_i))^2$ is minimized, where $T_i$ is the measured retention time of the $i$-th spectrum, and $t(S_i)$ the predicted retention time of the sequence $S_i$ identified in the $i$-th spectrum.

For example, we estimate the coefficients of the linear model by first computing the character frequency vector for each string in the dataset. The character frequency vector of a string is a vector of length $|\Sigma|$ that indicates how often a character occurs in the string. For example, the occurrence vector of the string AGA has value 2 at entry A, value 1 at entry G and value 0 at all other entries. Then, the retention time of a string $S$ is the product of its character frequency vector $\text{frq}(S)$ and the vector of the retention time coefficients $ct$. Standard software tools for statistical methods [79] can be used to compute $ct$, such that $\sum_i (T_i - \langle ct, \text{frq}(S_i) \rangle)^2$ is minimized.

We chose the tolerance parameter $\varepsilon$ independently for each prediction model by considering the difference between the measured and the predicted retention time of the sequences in the test set. Figure 3.13 shows the differences between the predicted and the measured retention times for all three models on the test dataset. We set $\varepsilon$ to half the difference between the maximum error $e_{\text{max}}$ and
the minimum error $e_{\text{min}}$, i.e. $\varepsilon = (e_{\text{max}} - e_{\text{min}})/2$. Concretely, we set $\varepsilon = 1000$ for the linear prediction model and $\varepsilon = 750$ for the position-dependent model.

The neighborhood-based prediction model has a very large predictive error for several sequences due to the small training dataset. Several coefficients are estimated based on few observations and others cannot be estimated at all. Therefore, we cannot extensively evaluate the identification rates of our algorithm with the neighborhood-based prediction model, as a much larger training dataset for estimat-
Figure 3.14: Number of spectra that are identified by DeNovoΔ, DeNovoΔLin, and DeNovoΔPos.

ing all parameters would be necessary. For our limited evaluation, we ignore the 5 largest and the 5 smallest retention time errors when picking the tolerance parameter and use $\varepsilon = 500$.

**Identification rates** We computed the all solutions with a score of at least 90% of the optimal score and a predicted retention time within the tolerance range for the 944 considered spectra using both DeNovoΔLin and DeNovoΔPos. Figure 3.14 shows the number of annotated sequences reported as best-scoring sequence by the three considered algorithms. While the majority of the spectra are either identified by all algorithms or not at all, 59 spectra are only identified when considering the retention time information.

Figure 3.15 shows a comparison of the identification rates with respect to the 5, 10, and 100 best-scoring of DeNovoΔ, DeNovoΔLin, and DeNovoΔPos. Without considering the retention time, the algorithm DeNovoΔ reported the annotated sequence as best-scoring
Figure 3.15: Position of annotated sequence in the list of reported sequences (sorted by score). DeNovoΔ reported the annotated sequence among the top 5 sequences in 73.4% of the spectra, DeNovoΔLin in 75.4% and DeNovoΔPos in 76.6% of the spectra.

sequence for 586 spectra (62.1%). Considering the linear retention time prediction model, DeNovoΔLin computed the annotated sequence with an optimal score for 610 spectra (64.6%). DeNovoΔPos considers the position-dependent prediction model and achieved the highest identification rate. The annotated sequence was reported as best-scoring sequence for 629 spectra (66.6%). The performance improvement decreases with increasing number of considered candidate sequences.

However, a filtering approach that considers the top 100 solutions
Figure 3.16: Identified spectra with respect to the length of the annotated sequence.

reported by \( \text{DeNovo}\Delta \), would not be as successful as the proposed algorithms. While the annotated sequence was reported by \( \text{DeNovo}\Delta \) for 793 spectra among the top 100 sequences, we identify the correct sequence with \( \text{DeNovo}\Delta\text{Lin} \) in 798 cases and with \( \text{DeNovo}\Delta\text{Pos} \) in 808 cases. Even an optimal filtering approach by retention time would miss the sequences that have not been reported by \( \text{DeNovo}\Delta \). For six spectra, \( \text{DeNovo}\Delta\text{Lin} \) and \( \text{DeNovo}\Delta\text{Pos} \) did not report the annotated sequence, where \( \text{DeNovo}\Delta \) did report it, as the predicted retention time of the annotated sequence was not in the chosen tolerance range.

The length of a peptide affects its retention time. The considered prediction models do not take into account the peptide’s length and
do not perform equally well on short and long peptides. Figure 3.16, shows a distribution of the number of identified spectra with respect to the length of the corresponding peptide sequence. DeNovoPoS shows the best performance for peptides with fewer than 14 amino acids. For longer peptides, the linear prediction model shows a superior identification rate on the considered dataset.

3.3.4 Discussion

An accurate retention time prediction model is crucial for exploiting the retention time information successfully. The identification rates of our algorithms depend on the choice of the tolerance parameter $\varepsilon$. Increasing $\varepsilon$ diminishes the effect of considering the retention time, while decreasing $\varepsilon$ might exclude the correct sequence from the search space.

In our evaluation, we considered a limited training dataset for estimating the retention time coefficients. While we have to estimate a small set of coefficients for our linear prediction model, the neighborhood-based prediction model has many retention time coefficients. Estimating these coefficients requires a large training dataset, as each coefficient needs to be estimated based on a sufficiently large set of observations. For a robust estimation of the coefficients of this model, we would require a much larger training set.

To get a glimpse of the performance of DeNovoNei, we set $\varepsilon = 500$ (in seconds) and analyzed the spectra from the test set, where the correct sequence was not excluded due to the predictive error. In three cases, the annotated sequence was reported by DeNovoNei, but by no other considered algorithm. The position of the correct
sequence improved compared to result of \texttt{DeNovo}△\texttt{Pos} for 12 spectra.

Our prediction models do not consider several other properties of a peptide that affect its retention time. For example, the length of a peptide has an influence on its retention time. More evolved prediction models \cite{53, 52} integrate a correction for the peptide length. Other approaches for retention time prediction, e.g. \cite{78, 77}, use support vector machines to predict the retention time of a peptide from several other aspects of a peptide’s sequence. The prediction models considered in this work is limited to “local” features of a sequence and cannot account for such features, such as the peptide length. However, as suggested in \cite{52}, a separate set of retention time coefficient can be estimated for short peptides in order to improve the prediction accuracy. This approach needs an even larger training dataset in order to accurately estimate the coefficients.

The running time of our prototypical implementations is in some cases not yet practical. \texttt{DeNovo}△\texttt{Lin} needs less than 3 seconds per spectrum for half of the considered spectra, but several hours in exceptional cases. In general, \texttt{DeNovo}△\texttt{Pos} is more time-consuming. Half of the spectra were analyzed within about 2 minutes each. The running time of our algorithm depends on the size of the spectrum graph. The algorithms considered two masses to be equal if they differ by at most 0.02 Dalton. Moreover, a simple merging algorithm is applied during the construction of the spectrum graph to reduce the size of the graph as described in \cite{90}. We observed a great variation of spectrum graph sizes in our experiments. The spectrum graphs contained roughly 8400 edges on average, whereas the largest observed graph contained 23000 edges. Spectra measured on low resolution lead to denser spectrum graphs, i.e. to a larger number of
edges, but a lower number of vertices. However, we did not study the performance and runtime of our algorithms on this type of spectra.

The proposed algorithms score sequences with a very simplistic scoring function that only counts explained and measured masses, but does not consider any other available information. For real-world applications, a more evolved scoring function using all available information needs to be integrated. While we introduce a new scoring model in Section 3.2, we explore ways of exploiting the retention time information in this section. The proposed algorithms open room for developing new scoring functions that consider both the retention time information and the symmetric difference scoring model.
3.4 Multiple Peptide De Novo Sequencing

In the previous sections, we assumed that the mass spectrometer acquires a mass spectrum of a single peptide. This assumption is very common for de novo sequencing algorithms, although real spectra are quite frequently mixture spectra, i.e. spectra containing fragment masses of multiple peptides. This is especially true for data-independent acquisition (DIA) experiments, where the mass spectrometer selects all peptides in a given mass range for simultaneous fragmentation. The resulting spectra are mixture spectra with fragment masses from multiple peptides.

In this section, we study the de novo peptide sequencing problem for mixture spectra from DIA experiments. We develop an algorithm that reconstructs the sequences of $k$ simultaneously analyzed peptides given the $k$ peptide masses and a fragmentation pattern $X$ containing the indistinguishable fragment masses of all $k$ peptides. The algorithm is also able to compute a set of suboptimal solutions (e.g. the 10 best-scoring solutions). This is, for example, of great interest to study local ambiguities in the results reported by a de novo sequencing algorithm.

Moreover, we propose a scoring function for time-resolved data that does not consider a single mass spectrum, but a series of mass spectra acquired over time. The scoring function is based on the assumption that the signal intensities of the peptide measured in MS$_1$ and its fragments measured in MS$_2$ are linearly dependent.
Problem Definition

We are given the masses of \( k \) peptides and a set \( X \) of fragment masses of all \( k \) peptides. We assume that the maximal mass difference of two peptide masses \( \Gamma := \max_{i,j} |M_i - M_j| \) is smaller than the mass \( \mu := \min_{a \in \Sigma} (m(a)) \) of the lightest character in the alphabet \( \Sigma \) (57 Da for amino acids). This assumption is reasonable for real experimental data, where the mass ranges for selecting peptides are usually smaller (e.g. 25 Da in SWATH MS [34]).

We focus on the problem of maximizing the number of explained masses rather than on minimizing the symmetric difference between explained and measured masses (Section 3.2.4). At the end of Section 3.4.2, we make some remarks on modifying the algorithm for minimizing the symmetric difference.

**Problem 4 (DeNovoMixSpectra).** Let \( \Sigma \) be an alphabet of characters with a mass \( m(a) \in \mathbb{N} \) for each \( a \in \Sigma \). Given \( k \) masses \( M_1, \ldots, M_k \) with a maximal mass difference \( \Gamma = \max_{i,j} |M_i - M_j| < \mu \) and a set \( X = \{ x_i \in \mathbb{N} \mid i = 1, \ldots, k \} \) of fragment masses, find a \( k \)-tuple of strings \( (S_1, \ldots, S_k) \) with \( m(S_i) = M_i \) for all \( i \in \{1, \ldots, k\} \) that maximizes the number of explained masses that are in \( X \), i.e.

\[
\left| X \cap \left( \bigcup_{i=1}^{k} \text{TS}(S_i) \right) \right|,
\]

among all \( k \)-tuples of strings \( (T_1, \ldots, T_k) \) with \( m(T_j) = M_j \) for all \( j \in \{1, \ldots, k\} \).
3.4.2 Algorithm

We present the main idea of the algorithm considering DeNovoMixSpectra with \( k = 1 \), i.e. the single peptide de novo sequencing problem. We construct a directed acyclic graph called extension graph. A solution for DeNovoMixSpectra corresponds to a longest path in this graph. The key difference of the extension graph and the spectrum graph [8] or the multigraph defined in Section 3.2.2 is, that we do not need to introduce forbidden pairs or variable edge or vertex weights to solve the de novo sequencing problem. This advantage comes at the cost of an increased number of vertices and edges compared to the spectrum graph.

After presenting the graph construction for our approach, we explain the difficulties when moving from \( k = 1 \) to \( k > 1 \) peptides. Finally, we present our algorithm for DeNovoMixSpectra and an arbitrary number of peptides.

Extension Graph for a Single Peptide

A common strategy of graph-based approaches to de novo sequencing is to construct a solution by extending two substrings, a prefix and a (reversed) suffix, until the sum of their masses is equal to the given mass \( M \) (cf. Section 3.2). In this way, one can account for the symmetry of the theoretical spectrum of a string and update the number of explained masses, respectively other additive scoring functions, correctly.

As introduced in Section 3.2, we consider the balanced extension rule for extending the two substrings. That is, we always extend the lighter of the two strings. Moreover, we do not consider an extension
if the sum of the substring masses after the extension exceeds the
given mass $M$. If we follow this rule, we can make two observations.
First, we will extend a substring to a mass larger than or equal to
$M/2$ at most once and only extend the other substring afterwards.
Second, let $A$ and $B$ be two substrings with $m(A) > m(B)$ resulting
from a sequence of balanced extensions. It holds that $m(B)$ is at least
as large as the mass of the heavier substring $A$ before its last extension
(cf. Lemma 1). Otherwise, we would not have extended that string
to $A$ in a previous extension by the balanced extension rule.

The balanced extension rule is useful, because it guarantees that
we can compute the set of newly explained masses of an extension
without knowing the complete character sequences of the two sub-
strings. For example, assume that we have a prefix of mass $x$ and a
reversed suffix of mass $y > x$ and that we extend the prefix by some
character to a prefix with mass $x' \geq x + \mu$. The set of newly explained
masses is $\{x', M - x'\} \setminus \{y, M - y\}$. To see that, first assume that
$x' \neq y$ and $x' \neq M - y$. Then, the masses $x'$ and $M - x'$ are newly
explained due to the second observation we made about the balanced
extension rule in the previous paragraph. Note that this also holds if
$x' > M/2$, as we know that $x' + y \leq M$. On the other hand, assume
that $x' = y$ or $x' = M - y$ (Figure 3.17). In this case, no mass is
newly explained by this extension, as $y$ and $M - y$ have already been
explained before this extension. Hence, the set of newly explained
masses does not depend on the complete character sequences of the
prefix and the suffix.

**Extension graph** We model the space of all possible extension
steps that lead to a pair of substrings with total mass $M$ in a directed
acyclic graph termed *extension graph*. Every vertex in this graph
represents a pair of two masses. An edge corresponds to a balanced extension step and its weight is the number masses in $X$ that are newly explained by this extension step. A sequence of balanced extension steps is represented by a path in the extension graph. The problem of finding a string of mass $M$ that explains a maximal number of masses in $X$ is equivalent to finding a heaviest path in this graph starting at the vertex representing the pair of masses $(0, 0)$ and ending at a vertex whose masses sum up to $M$. Lu et al. [57] propose a similar graph model, but consider a different extension rule.

Formally, the vertex set of the extension graph $G(X, M)$ consists of every pair of two masses in $X_M := \{m, M-m \mid m \in X\}$. Note that $X_M$ is exactly the vertex set of the multigraph defined in Section 3.2.2. The edge set of the extension graph represents all possible balanced extension steps. There exists an edge from a vertex $(x, y)$ to a vertex $(x, y')$, iff $x > y$ (balanced extension rule), $y' > y$, and if $y' - y$ is the mass of some string of characters from $\Sigma$. Similarly, there is an edge from a vertex $(v, w)$ to a vertex $(v', w)$, iff $v \leq w$, $v' > v$ and if there exists a string of mass $v' - v$. Note that the edge from $(v, w)$ to $(v', w)$ represents all extensions by one or multiple characters with mass

![Figure 3.17](image-url)
\( v' - v \). In particular, if the edge represents an extension by multiple characters, it also represents the extension by any permutation of these characters. This is in contrast to the multigraph in Section 3.2.2, where every permutation is represented by its own edge. It is not necessary to add multiple edges between the same set of vertices here, as we consider maximizing the number of explained masses in \( X \) rather than minimizing the symmetric difference.

The weight of an edge corresponds to the number of newly explained masses by the corresponding extension step that are in \( X \). We have seen that the newly explained masses can be computed without knowing the character sequence of the substrings. In the same way, we can define the weight of an edge from \((x, y)\) to \((x', y)\) as the number of explained masses by an extension step represented by this edge that are in \( X \):

\[
\text{weight}((x, y), (x', y)) := \left| \left( \{x', M - x'\} \setminus \{y, M - y\} \right) \cap X \right|. \tag{3.21}
\]

The weight of a path in this graph is the sum the weights of its edges.

The weight of a heaviest path starting at vertex \((0, 0)\) and ending at vertex \((x, y)\) corresponds to the maximal number of masses in \( X \) that can be explained by any prefix with mass \( x \) and any suffix with mass \( y \) of a string with mass \( M \). This also holds if we consider edges that represent extensions by multiple characters, despite the fact that we only consider the masses represented by the vertices of the graph in the definition of the edge weights. An extension by multiple characters can explain additional masses that are not considered in Definition 3.21. In this case, there exists a different path using more edges that additionally considers these masses and has a higher weight.
We can compute the maximal number of masses in $X$ that can be explained by a string of mass $M$ by finding a heaviest path starting at vertex $(0,0)$ and ending in a vertex $(x,M-x)$ with $x \in X_M$. Computing such a path in a directed acyclic graph is possible with a standard dynamic program with a time complexity in $O(|V| + |E|)$ [13], where $V$ is the vertex set and $E$ the edge set of the extension graph. The extension graph has $O(|X|^2)$ vertices and each vertex has $O(|X|)$ outgoing edges. Therefore, we can compute a heaviest path in the extension graph in $O(|X|^3)$ running time.

**From $k = 1$ to $k = 2$ Peptides**

An evident idea for extending the above approach to two peptides is to iteratively extend a prefix and a suffix for each of the two strings $S_1$ (with mass $M_1$) and $S_2$ (with mass $M_2$). However, we have to consider two issues while simultaneously extending the four substrings. First, we cannot construct every possible 4-tuple of strings with masses $M_1$ and $M_2$ if we always extend the lightest of the four substrings. For example, assume that we have computed two substrings $A$ and $B$ with $m(A) + m(B) = M_1$ for the first string $S_1$ and two substrings $C$ and $D$ with $m(C) + m(D) < M_2$ for the second mass $M_2$. If $A$ is the only substring with smallest mass among the four substrings $A, B, C, and D$, we cannot extend $C$ or $D$ anymore. Hence, we cannot complete the construction of the second string $S_2$. Second, we have to consider two theoretical spectra with different symmetries, which complicates the computation of the set of newly explained masses of an extension step.
Balanced extension rule  To consider the first issue, we weaken the balanced extension rule by ignoring a pair of substrings if their combined mass equals the corresponding peptide mass $M_i$. We will see that we are still able to compute the set of newly explained masses without knowing the character sequences of the four substrings. The balanced extension rule for two peptides always extends the lightest substring, but ignores a pair of prefix and suffix strings if their combined mass is equal to the corresponding peptide mass. We only consider extensions, where the sum of the prefix mass and suffix mass of the $i$-th peptide is smaller than the mass $M_i$. With this modified extension rule we can compute any pair of strings with mass $M_1$, respectively $M_2$. In the above example, we can extend one of the substrings $C$ and $D$, as we do not consider $A$ and $B$ with $m(A) + m(B) = M_1$ in the balanced extension rule.

Newly explained masses  When we are computing the set of masses that are newly explained by an extension step, we have to consider two theoretical spectra with different symmetries. Fortunately, we can still compute this set without knowledge of the complete character sequences of the four substrings if the difference of the given masses $M_1$ and $M_2$ is smaller than $\mu$. Assume that we are given the set $X$ and two masses $M_1$ and $M_2$ and that we have already computed two substrings with masses $a$ and $b$ for $M_1$ and two substrings with masses $c$ and $d$ for $M_2$ following our balanced extension rule. We note that $a + b \leq M_1$ and $c + d \leq M_2$. Let $a$ be a smallest mass among $a, b, c,$ and $d$. Assume that we extend the substring with mass $a$ by one character to a substring of mass $a'$. The following lemma shows that we can compute the number of newly explained masses by this extension step by only considering the masses $a, b, c, d,$ and $a'$. 

91
Lemma 2. Given two masses $M_1, M_2$ and four substrings $A, B, C, D$ with $m(A) + m(B) < M_1$ and $m(C) + m(D) \leq M_2$ resulting from a sequence of balanced extensions, where $m(A)$ is the lightest among the four substrings. The newly explained masses by extending $A$ by one character in $\Sigma$ to a string $A'$ are

$$\{m(A'), M_1 - m(A')\} \setminus \{m(B), m(C), m(D)\}$$

$$M_1 - m(B), M_2 - m(C), M_2 - m(D)\}.$$ 

Proof. We know that $m(A')$ and $M_1 - m(A')$ are not explained by any prefix of $A$ or $B$ (including the complete strings $A$ and $B$), unless $m(A') = m(B)$, due to the properties of the balanced extension rule.

We denote the largest proper prefix of a string $T = t_1 \ldots t_n$ by $T^- := t_1 \ldots t_{n-1}$. To see that $m(A')$ and $M_1 - m(A')$ are not explained by prefixes of $C$ and $D$ (including the complete strings $C$ and $D$) unless $m(A') = m(C)$ or $m(A') = m(D)$, we note that $m(C^-)$ and $m(D^-)$ are both at most as large as $m(A)$ due to the balanced extension rule. Hence, the difference between $m(A)$ and $m(C^-)$, respectively $m(A)$ and $m(D^-)$, is at least $\mu$.

Let us consider the masses $M_2 - m(C^-)$ and $M_2 - m(D^-)$. The mass difference of $M_1 - m(A')$ and each of these masses is at least $\mu - |M_1 - M_2| > 0$: We know that $m(A') - \max(m(C^-), m(D^-)) \geq \mu$ and this difference is reduced by at most $|M_1 - M_2|$ if we consider the differences of the complementary masses $M_1 - m(A')$ and $M_2 - m(C^-)$, respectively $M_1 - m(A')$ and $M_2 - m(D^-)$.

Finally, we consider the difference between $m(A')$ and $M_2 - m(C^-)$,
respectively between $m(A')$ and $M_2 - m(D^-)$. We first note that

$$m(A') + m(B) \leq M_1$$
$$m(A') + m(A) \leq M_1$$
$$0 \leq M_1 - m(A') - m(A).$$

We distinguish two cases: First, if $M_1 = m(A') + m(A)$, it follows that $m(B) = M_1 - m(A)$ and that the set of newly explained masses is empty. Second, if $M_1 > m(A') + m(A)$, we only have to consider cases where $M_1 - m(A') - m(A) \geq \mu$, because $A'$ and $B$ cannot be extended to substrings with masses that sum up to $M_1$ if $0 < M_1 - m(A') - m(A) < \mu$. As $\max(m(C^-), m(D^-)) \leq m(A)$, we get $M_2 - \max(m(C^-), m(D^-)) - m(A') \geq \mu - |M_1 - M_2|$. By symmetry, the same holds for the difference between $M_1 - m(A')$ and $m(C^-)$, respectively $M_1 - m(A')$ and $m(D^-)$. Therefore, $m(A')$ and $M_1 - m(A')$ cannot be explained by any proper prefix of $C$ and $D$.

**Extension graph** We define the extension graph $G(X, M_1, M_2)$ for $k = 2$ peptides as follows. The vertex set of the graph is a subset of $(X_{M_1})^2 \times (X_{M_2})^2$, where $X_M := \bigcup_{m \in X} \{m, M - m\}$. A vertex is a 4-tuple of masses, such that the sum of the first two masses is at most $M_1$ and the sum of the remaining two masses is at most $M_2$.

We define two types of edges. Type-1 edges represent extension steps, where the lightest among the four substrings is extended: A vertex $(a, b, c, d)$ has an outgoing type-1 edge to a vertex $(a', b, c, d)$, if $a = \min(a, b, c, d)$ and if there exists a string with mass $a' - a$. Outgoing type-1 edges of vertices with a minimal mass at another position in the 4-tuple are defined analogously. Type-2 edges represent extension steps, where either the first two masses in the 4-tuple or
the last to masses sum up to $M_1$, respectively $M_2$, and the lighter of the remaining masses is considered for the next extension. A vertex $(v, w, x, y)$ has an outgoing type-2 edge to a vertex $(v, w, x, y')$, if $v + w = M_1$, $y \leq x$, and if there exists a string with mass $y' - y$. The weight of an edge (of any type) from a vertex $(a, b, c, d)$ to a vertex $(a', b, c, d)$ is defined as the number of masses in $X$ that are newly explained masses by a corresponding extension step:

$$weight((a, b, c, d), (a', b, c, d)) := \left| \left( \{a', M_1 - a'\} \setminus \{b, M_1 - b, c, M_2 - c, d, M_2 - d\} \right) \cap X \right|. \quad (3.22)$$

A heaviest path starting at vertex $(0, 0, 0, 0)$ and ending in a vertex $(x, M_1 - x, y, M_2 - y)$ for some $x \in X_{M_1}$ and some $y \in X_{M_2}$ represents a sequence of extension steps of two pairs of substrings, where the combined mass of one pair is $M_1$ and the combined mass of the other pair is $M_2$, that maximize the number of explained masses in $X$ among all such pairs of substrings. The extension graph has $O(|X|^4)$ vertices and each vertex has at most $|X|$ outgoing edges. We can compute such a heaviest path in $O(|X|^5)$ running time [13]. Having such a heaviest path, we can easily reconstruct two strings with masses $M_1$ and $M_2$ that maximize the number of explained masses in $X$.

**De Novo Sequencing for $k > 2$ Peptides**

Our construction of the extension graph can easily be extended for solving DeNovoMixSpectra with an arbitrary number $k$ of peptides. In the general case, the vertex set $V$ of the extension graph is a subset of $(X_{M_1})^2 \times \cdots \times (X_{M_k})^2$. A vertex $(a_1, b_1, a_2, b_2, \ldots, a_k, b_k)$ represents a set of $k$ pairs of substrings with masses $a_i$ and $b_i$, such
that $a_i + b_i \leq M_i$. The edge set $E$ is defined analogously to the balanced extension rule as defined for the case of $k = 2$ above. That is, we ignore prefixes and suffixes with a combined mass equal to the corresponding peptide mass. In this way, the extension graph represents all possible sequences of extension steps starting from the set of $2k$ empty substrings. We define the weight of an edge $e$ from a vertex $(a_1, \ldots, a_i, \ldots, b_k)$ to a vertex $(a_1, \ldots, a_i', \ldots, b_k)$ as

$$\text{weight}(e) := \left| \left\{ a_i', M_i - a_i' \right\} \setminus \bigcup_{j \in \{1, \ldots, k\}} \left\{ a_j, M_j - a_j, b_j, M_j - b_j \right\} \right| \cap X.$$  

(3.23)

The weight of an edge is exactly the number of masses in $X$ that are newly explained by the extension step represented by this edge.

The extension graph has $O(|X|^{2k})$ vertices and every vertex has at most $|X|$ outgoing edges. The size of the edge set is in $O(|X|^{2k+1})$. Consequently, a solution for DeNovoMixSpectra can be computed in time $O(|X|^{2k+1})$.

**Remark on the Symmetric Difference Scoring Model**

A similar approach can be used if we want to consider the symmetric difference scoring model introduced in Section 3.2. However, it is not sufficient that a vertex in the extension graph represents a $2k$-tupel of masses. This is due to the fact, that we need to consider the last extension for each substring to compute the score of a balanced
extension in this model. Instead of an extension graph, we would therefore need to construct an extension multigraph. A vertex in this multigraph has to represent the mass of the lightest substring (that will be chosen for the next extension) and $2(k - 1)$ edges of the spectrum multigraph defined in Section 3.2. Moreover, an edge in the extension multigraph represents a balanced extension with a single string (rather than all extensions with all possible strings with a specific mass).

### 3.4.3 Scoring Function for Time-resolved Data

In the previous section, we considered the problem of finding a $k$-tuple of strings with masses $M_1, \ldots, M_k$ that maximizes the number of explained masses in $X$ among all such $k$-tuples. This is obviously a very simplistic scoring function. In this section, we aim for a scoring function for time-resolved data obtained from experiments in data-independent acquisition mode. In such experiments, the mass spectrometer acquires a mass spectrum of all peptides in a specific mass range every few seconds. Thus, we are not only given a set of masses $X$, but additionally a vector of signal intensities measured over time for each of these masses.

The key idea of our scoring function is to relate the signal intensities of the peptide and its fragments over time and measure their similarities. The intensity of a signal in the spectrum corresponds to the abundance of the measured molecule. The abundances of a peptide’s fragments depend on the abundance of the peptide itself. The probability of a fragmentation event is influenced by various, partially unknown factors. However, we can assume that this probability does not change over time and, hence, that the signal intensities
of a peptide and the signal intensities of its fragments are linearly dependent.

We propose to not only consider the number of explained masses, but also the similarity of the signal intensities of the explained masses with the signal intensities of the corresponding peptide. We denote the signal intensity vector of a mass \( m \in X \) by \( I(m) \in \mathbb{R}^t \). All intensity vectors have the same length \( t > 0 \) corresponding to the number of spectra that have been acquired for the considered peptides. Intensities that are at the same position in two vectors always correspond to signal intensities measured in the same measurement cycle of the experiment. We define the similarity of two signal intensity vectors \( I(m) \) and \( I(M_i) \) by the minimum least squares error

\[
\min_{0 \leq \alpha \leq 1} \sum_{j=1}^{t} \left( (I(m) - \alpha \cdot I(M_i))^2 \right)
\]

We are interested in a \( k \)-tuple of strings with masses \( M_1, \ldots, M_k \) that minimizes the total least squares error between the signal intensities of the masses in \( X \), that are explained by the strings, and the signal intensities of the peptides measured in MS1. That is, we want to compute \( k \) strings \( S_1, \ldots, S_k \) with \( m(S_i) = M_i \) for \( i = 1, \ldots, k \) that minimize the total least squares error

\[
\sum_{m \in X} \left( \min_{0 \leq \alpha_{im} \leq 1} \sum_{k=1}^{t} \left( I(m) - \alpha_{im} \cdot I(M_i) \right)^2 \right),
\]

where \( \alpha_{im} \) is the scaling factor for \( m \in TS(S_i) \).

In this problem variant, we have to distinguish whether a mass \( m \in X \) is explained by a single string or by multiple strings. Note that this was not the case when maximizing the number of explained masses in \( X \), as it is not relevant by which string and by how many
strings the masses are explained if we only want to count them. The scaling factors \( a_{im} \) for \( I(M_i) \) and \( a_{jm} \) for \( I(M_j) \) cannot be chosen independently, but the choice of one influences the choice of the other one. However, we can still reduce this problem variant to the problem of finding a heaviest path in the extension graph by defining appropriate edge weights.

We know from the previous section that we can compute the set of newly explained masses of an extension step by only considering the masses of the substrings before and after the extension step. We can additionally compute, which strings already explain some mass that is explained by an extension step, but not newly explained. The scaling factors of these masses (but no other scaling factors) are affected by this extension step. We define edge weights in the extension graph, such that the weight of an edge indicates by how much the extension step decreases the total least squares error of the solution. Consider a vertex \( v = (a_1, b_1, a_2, b_2, \ldots, a_k, b_k) \) in the extension graph. The set of vertex masses of a vertex of \( v \),

\[
X_v = \bigcup_{i \in \{1, \ldots, k\}} \{a_i, M_i - a_i, b_i, M_i - b_i\},
\]

is the set of masses of the vertex and their complementary masses. We define the set \( X_v(m) \) that indicates, which masses in \( X_v \) explain a mass \( m \). An index \( i \in \{1, \ldots, k\} \) is in \( X_v(m) \) if and only if \( m \) is equal to one of the masses in \( \{a_i, M_i - a_i, b_i, M_i - b_i\} \subset X_v \). The least squares error of the vertex masses \( X_v \) is
Let \( e \) be an edge from vertex \( v = (a_1, \ldots, a_i, \ldots, b_k) \) to some vertex \( w = (a_1, \ldots, a'_i, \ldots, b_k) \). At this edge \( e \), the total least squares error is reduced by the difference of the least squares error of the vertex masses of \( w \) and the vertex masses of \( v \) without \( a_i \) and \( M_i - a_i \). We define the reduction of the least squares error of edge \( e \) by \( \text{weight}(e) = f(X_w \setminus \{a_i, M_i - a_i\}) - f(X_v \setminus \{a_i, M_i - a_i\}) \). Thus, we compare the least squares error of the vertex masses of \( w \) with the least squares error of the vertex masses \( X_v \) without \( a_i, M_i - a_i \). We do not consider \( a_i \) and \( M_i - a_i \), as these masses are not in \( X_w \) and their scaling factors are not affected by the extension step. The two terms \( f(X_w \setminus \{a_i, M_i - a_i\}) \) and \( f(X_v \setminus \{a_i, M_i - a_i\}) \) only differ in the terms for the masses that are equal to \( a'_i \) or \( M_i - a'_i \). The difference of these terms corresponds to the reduction of the total least squares error by this extension.

We can compute a solution for DeNovoMixSpectra with this scoring function using the techniques described in the previous section, but considering the above definition of the edge weights and computing the lightest instead of the heaviest path, as we consider a minimization problem.

### 3.4.4 Experimental Results

In this section, we study the performance of our algorithm on some exemplary cases of mixture spectra from the SWATH MS Gold Stan-
standard (SGS) dataset [80]. We implemented our algorithm in C++ and used the output from DIA-Umpire [92] and the measured spectra as input for our algorithm. In several cases, the results improved when considering $k = 2$ peptides instead of a single peptide. However, a rigorous evaluation and comparison with other de novo sequencing tools is necessary to assess the quality of the results computed by our algorithm.

**Dataset** We considered experimental data from 422 synthesized peptides measured in three proteomic backgrounds (water, yeast, and human) and with various dilution levels [80]. We focused on the data obtained at highest dilution level in yeast background.

We use the DIA-Umpire analysis workflow [92] to first identify the synthesized peptides in the data. DIA-Umpire first detects the signals of peptides and fragments in the MS$_1$ and MS$_2$ spectra. Then, fragment signals are assigned to peptide signals based on the similarity of the signal intensities measured over time. DIA-Umpire assigns fragment signals to multiple peptides in uncertain cases. Finally, DIA-Umpire generates a so-called pseudo-spectrum for every detected peptide. A pseudo-spectrum for a peptide detected in MS$_1$ contains all fragment signals that were assigned to this peptide in the previous step. Pseudo-spectra are assumed to contain no fragment signals of other co-eluting peptides and are used as input for traditional tools for peptide identification. Every pseudo-spectrum is assigned to one of three quality tiers based on the quality of the peptide signal. We refer to [92] for a detailed description of this assignment.

We considered only pseudo-spectra of highest quality (quality tier 1, Q1) and analyzed all generated Q1 pseudo-spectra for yeast background with Comet using a database containing all synthesized
peptides. We validated the results using the Trans-Proteomic Pipeline (TPP, [46]) and considered the results as reference annotations for our comparison. We then selected a set of exemplary spectra of synthesized peptides using the following criteria: First, we only considered spectra containing fragments of the synthesized peptide and at least one other, simultaneously analyzed peptide. We only consider spectra, where a Q1 pseudo-spectrum at charge state 2 was generated for the synthesized peptide and for at least one additional peptide. Moreover, we inspected the pseudo-spectra and did not consider those, where two or more consecutive prefix and suffix masses were missing in the pseudo-spectrum of the synthesized peptide. The sequence of these peptides can only be reconstructed using spectrum graphs with edges representing three or more characters. However, considering these edges results in very large extension graphs. With these criteria, we selected 27 example spectra from the dataset.

Assessment measure We analyzed the performance of our algorithm by comparing the best-scoring sequence reported by the algorithm with the known sequence of the synthesized peptide. We considered the problem DeNovoMixSpectra both for $k=1$ peptide and $k=2$ peptides. In the latter case, we considered the synthesized peptide in combination with every other peptide with a Q1 pseudo-spectrum generated by DIA-Umpire.

We compared a sequence reported by our algorithm with the known sequence of the synthesized peptide by considering their sets of prefix masses: The recall of a reported sequence is the number of prefix masses it has in common with the known sequence divided by the number of prefix masses of the known sequences (Equation (3.13)).
Comparison of identification rates  For each of the 27 synthesized peptides, we first computed the best-scoring sequence for DeNovoMixSpectra with $k = 1$. Our algorithm requires three inputs: the mass $M$ of the peptide, the set $X$ of fragment masses, and a set of consecutively measured spectra for our time-resolved scoring approach. We used the mass of the peptide as detected by DIA-Umpire in the yeast background measurements as the first input and the masses from the pseudo-spectrum generated by DIA-Umpire from the yeast background measurements as the second input (i.e. the set $X$). As the third input, we used the centroided MS$_1$ and MS$_2$ spectra measured in yeast background. We considered all measured spectra in the elution time range of the peptide as detected by DIA-Umpire (usually around 40 to 60 seconds). In our scoring approach, we focused on the most prominent fragment types, the $b$- and $y$-ions that result from a fragmentation at peptide bonds, and did not consider additional types.

The results of this first experimental evaluation is shown in the second column of Table 3.1. In four cases (15%), the best-scoring sequence computed by our algorithm contained all prefix masses of the true sequence. In ten cases (37%), the best-scoring sequence still contained 75% of the prefix masses of the known sequence.

Then, we studied the best-scoring sequence considering one additional co-eluting peptide, i.e. we solved DeNovoMixSpectra for $k = 2$. For every synthesized peptide, we considered every co-eluting peptide with a Q$_1$ pseudo-spectrum. Our algorithm requires the following three inputs. The peptide masses $M_1$ and $M_2$ were used as detected by DIA-Umpire. For the set $X$ of fragment masses, we did not consider the union of both pseudo-spectra generated by DIA-Umpire. Instead, we considered both pseudo-spectra independently.
in order to reduce the size of the extension graph. The vertex set of the extension graph was computed as a subset of \((X_1)_{M_1}^2 \times (X_2)_{M_2}^2\), where \(X_1\) and \(X_2\) are the sets of masses from the two pseudo-spectra generated by DIA-Umpire. In this way, fragment masses, that have been uniquely assigned to one peptide by DIA-Umpire, can only be used for this peptide by our algorithm. On the other hand, the algorithm can use a fragment mass for both peptides, if DIA-Umpire was not able to uniquely assign it. In this way, we accounted for the partial separation performed by DIA-Umpire without assuming that the pseudo-spectra only contain fragments of a single peptide. It would of course be possible to use the union of both pseudo-spectra as input set \(X\) or to ignore the preprocessing of DIA-Umpire at all. However, this would increase the running time of our algorithm that is in the range of several hours in some cases.

The results of this second experiment are summarized in the third and forth column of Table 3.1. In 13 cases (48%) the recall of the best scoring sequence improved compared to the best-scoring sequence when considering the 1-peptide de novo sequencing problem. Our algorithm was able to reconstruct at least 75% of the prefix masses for seven (26%) peptides considering a co-eluting peptide, where this was not possible considering only the synthesized peptide.

We took a closer look at all cases where the results improved by at least 25% when considering a co-eluting peptide (8 cases) and made the following observation. Besides one case (GSDYSEILDK), the peak retention time (i.e. the retention time when the highest signal intensity was measured) of the synthesized peptide and the co-eluting peptide differed by at most 5 seconds. Consider for example Figure 3.18. DIA-Umpire detected the synthesized peptide HGVAI.TTTK
(m/z ratio 503.79 in MS1) and three co-eluting peptides. The figure shows the elution profile of each of these peptides and in the upper left corner the retention time intervals. The best-scoring sequence of our algorithm without considering co-eluting peptides had a recall of only 30%. However, when considering the co-eluting peptide with m/z ratio 501.74 while reconstructing the sequence, the recall of the best-scoring sequence improved to 80%.

Figure 3.18: Elution profile of HGVAAITTTK and three co-eluting peptides detected by DIA-Umpire. The elution time intervals as estimated by DIA-Umpire are depicted in the upper left corner (elution profile of HGVAAITTTK in red).

While these experiments indicate that our algorithm can improve identification rates especially if preprocessing mixture spectra into
multiple pseudo-spectra is difficult, a rigorous evaluation and a comparison to other de novo sequencing algorithms is necessary to assess the performance of our algorithm. Moreover, we would like to note that the running time of our algorithms depends heavily on the size of the considered spectra and the resulting extension graph. Our prototypical implementation is therefore not yet suitable for a practical application at large scale.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>recall $k = 1$</th>
<th>recall $k = 2$</th>
<th>co-eluting peptides</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEVAALAAENK</td>
<td>100%</td>
<td>100%</td>
<td>+0%</td>
</tr>
<tr>
<td>APIYGTQNVYEVLDK</td>
<td>100%</td>
<td>100%</td>
<td>+0%</td>
</tr>
<tr>
<td>HAEILSEK</td>
<td>100%</td>
<td>100%</td>
<td>+0%</td>
</tr>
<tr>
<td>IVTEAVEIEQR</td>
<td>100%</td>
<td>100%</td>
<td>+0%</td>
</tr>
<tr>
<td>DLGPALADSSHDVK</td>
<td>93%</td>
<td>93%</td>
<td>+0%</td>
</tr>
<tr>
<td>FGTEIITETVSK</td>
<td>92%</td>
<td>92%</td>
<td>+0%</td>
</tr>
<tr>
<td>EYTEGVNGQPSIR</td>
<td>85%</td>
<td>85%</td>
<td>+0%</td>
</tr>
<tr>
<td>ADTLTAQELQQFK</td>
<td>85%</td>
<td>85%</td>
<td>+0%</td>
</tr>
<tr>
<td>SSSESFGYDAESLK</td>
<td>79%</td>
<td>79%</td>
<td>+0%</td>
</tr>
<tr>
<td>HFISELEK</td>
<td>75%</td>
<td>75%</td>
<td>+0%</td>
</tr>
<tr>
<td>INPSQQLQQQLNR</td>
<td>69%</td>
<td>77%</td>
<td>+8%</td>
</tr>
<tr>
<td>LWGATIGDQSMELR</td>
<td>64%</td>
<td>79%</td>
<td>+14%</td>
</tr>
<tr>
<td>LTILEELR</td>
<td>63%</td>
<td>100%</td>
<td>+37%</td>
</tr>
<tr>
<td>ELASGLSFPVGFK</td>
<td>62%</td>
<td>69%</td>
<td>+7%</td>
</tr>
<tr>
<td>GDLAQILQLTR</td>
<td>55%</td>
<td>64%</td>
<td>+9%</td>
</tr>
<tr>
<td>Sequence</td>
<td>recall $k = 1$</td>
<td>recall $k = 2$</td>
<td>co-eluting peptides</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>GILFVGSGVSGGEEGAR</td>
<td>53%</td>
<td>53%</td>
<td>+0%</td>
</tr>
<tr>
<td>ANLIPVIAK</td>
<td>44%</td>
<td>44%</td>
<td>+0%</td>
</tr>
<tr>
<td>YGGHSMSDPGTTYR</td>
<td>36%</td>
<td>36%</td>
<td>+0%</td>
</tr>
<tr>
<td>GISNEGQNASK</td>
<td>33%</td>
<td>83%</td>
<td>+50%</td>
</tr>
<tr>
<td>VNPETGIIDYDTEK</td>
<td>33%</td>
<td>80%</td>
<td>+47%</td>
</tr>
<tr>
<td>HGVAAITTTK</td>
<td>30%</td>
<td>80%</td>
<td>+50%</td>
</tr>
<tr>
<td>SDNLSQHIK</td>
<td>22%</td>
<td>67%</td>
<td>+45%</td>
</tr>
<tr>
<td>LVEDPQVIAPFLGK</td>
<td>21%</td>
<td>29%</td>
<td>+8%</td>
</tr>
<tr>
<td>GAIAAAHYIR</td>
<td>20%</td>
<td>50%</td>
<td>+30%</td>
</tr>
<tr>
<td>SYGAQVDVVRPLK</td>
<td>8%</td>
<td>8%</td>
<td>+0%</td>
</tr>
<tr>
<td>ELGQSGVDTYLQTK</td>
<td>7%</td>
<td>86%</td>
<td>+79%</td>
</tr>
<tr>
<td>GSDYSEILDK</td>
<td>0%</td>
<td>30%</td>
<td>+30%</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of the results for the 27 analyzed peptides. The recall of the best-scoring sequence is reported both when considering only the peptide of interest ($k = 1$) and when considering the peptide of interest in combination with every co-eluting peptide with a $Q_1$ pseudo-spectrum ($k = 2$, only the best result is shown).
Peptide identification by database search is a reliable approach for identifying known peptides via tandem mass spectrometry. The measured fragmentation pattern is compared with the expected fragmentation patterns of all sequences in a protein sequence database to find the sequence in the database with the most similar expected fragmentation pattern. To speed up this analysis, database filtration techniques quickly discard sequences without similar fragment masses.

In this chapter, we consider database filtration by a sequence of masses called \textit{blocked pattern}. A blocked pattern with $m$ masses matches a sequence if the sequence can be partitioned into $m$ consecutive subsequences such that the mass of the $i$-th subsequence is
equal to the $i$-th mass in the pattern for all $i = 1, \ldots, m$. The *blocked pattern matching (BPM)* problem is to find all sequences in a database that match a given blocked pattern.

In the first section of this chapter, we discuss related work on database filtration in the context of peptide identification. In Section 4.2, we define the (exact) BPM problem and two variants of approximate BPM. The first variant accounts for post-translational modifications that are common in practice and affect the masses of the peptide and its fragments. The second variant allows for a single mutation, i.e. the insertion, deletion, or replacement of one character in a sequence, such that it matches the given blocked pattern. Accounting for modifications and mutations is important for many practical applications. In Section 4.3, we present a memory-efficient algorithm that uses multiple priority search trees to query the database efficiently. This approach is able to solve the approximate blocked pattern matching variants, as we discuss in Section 4.4. Finally, we evaluate the proposed algorithms in Section 4.5 using synthetic patterns and patterns generated from real spectra. In an application on patterns generated from spectra of different instrument types, modification-tolerant blocked pattern matching increases the rate of identifiable spectra by 17% compared to exact blocked pattern matching at a small computational cost.

### 4.1 Related Work

With the rapidly growing size of databases, querying a database by a simple linear sweep becomes prohibitively slow. In classical string matching problems, indexing techniques can be successfully applied
to achieve query times that are independent of the size of the database and only depend on the length of the query. Unfortunately, proteomics lacks such a time and space efficient indexing technique for matching mass spectra against sequences. Therefore, it is common to speed up database search approaches using database filtration techniques. The idea is to first exclude large parts of the database that are clearly not of interest for a given query mass spectrum. The remaining small subset of the database is then considered in the expensive search phase.

A very common filtering criterion is the peptide’s mass \([27, 75, 14]\). In this approach, the mass spectrum is only matched against sequences with a mass that is similar to the measured peptide mass. Several algorithms for finding sequences in a database with a given mass have been proposed \([23, 10, 11, 70]\). However, this approach is not effective enough for huge databases and it is not suitable for peptides with mutations or post-translational modifications.

Mann and Wilm \([62]\) pioneered an approach that uses partial de novo sequencing solutions for filtering the database. First, sequence tags, small subsequences consisting of a few amino acid characters, are reconstructed de novo. Then, only sequences matching these sequence tags are considered in the database search. Such approaches \([88, 41, 31, 55]\) are fast due to a very high filtration efficiency, but they are only successful if a correct sequence tag can be computed reliably, which turned out to be a challenging task.

Kim et al. \([48]\) address this problem by computing a covering set of de novo sequences (spectral dictionary) that contains the correct sequence with high probability. Spectral dictionaries tend to grow very fast with the length of the analyzed peptide. The approach was further improved \([44]\) to gapped spectral dictionaries, which are cov-
ering sets of so-called gapped peptides. A gapped peptide is a de novo sequence, where subsequences that cannot be reliably reconstructed are replaced by the corresponding mass. For example, assume that we analyze a peptide with sequence AGTCA. If the instrument neither measures the mass of the prefix AG nor of the suffix TCA, the sequences AGTCA and ATGCA cannot be distinguished from the mass spectrum. The gapped peptide $A[158]CA$ (where 158 is the mass of the subsequence GT) accounts for this ambiguity.

Filtering a database by gapped peptides is formulated as the so-called blocked pattern matching (BPM) problem [70]. An amino acid sequence is a string over an alphabet, where each character is assigned its own mass. The mass of a string is the sum of the masses of its characters. A gapped peptide is transformed into a sequence of masses called (blocked) pattern by replacing all amino acid characters by their corresponding mass. For example, the gapped peptide $A[158]CA$ is transformed into the pattern $(71, 158, 103, 71)$. A string matches the pattern, if the sum of the first $i$ masses is equal to the mass of some prefix of the string for every choice of $i$. The BPM problem is to find all substrings in a long string $S$ that match a given pattern $P$, where $S$ is the concatenation of all amino acid sequences in the database.

The BPM problem is different from classical string matching problems, as it focuses on matching the masses of substrings and as the mass of a string is invariant under permutations. Many powerful techniques, such as for example suffix trees, do not directly lead to a satisfying solution for the BPM problem. BPM algorithms can be divided into two phases: a preprocessing phase, where only the string $S$ is known and an index data structure is constructed, and a query phase, where a pattern $P$ needs to be matched against the
substrings of $S$. We usually expect many queries on the same string $S$ and are interested in a fast query phase, while we are willing to invest additional time in the preprocessing phase. In the context of memory efficiency, we mainly focus on the preprocessing phase and on the size of the index data structure.

An evident characteristic of the BPM problem is that patterns and strings cannot be directly compared in a trivial way. At first sight, we can either compute all possible patterns of the substrings of $S$ and compare them with $P$ in the query phase, or we compute all strings matching the query pattern and search for each of them in $S$. The first option leads to a pattern-based index data structure, while the second option leads to a string-based index data structure. We can distinguish the previously proposed algorithms based on the type of their index data structure.

Ng, Amir, and Pevzner [70] propose a pattern-based approach, where $S$ is transformed into a trie of patterns. This leads to a query time in $O(m)$ for a pattern of length $m$, while the time and space for constructing the index can grow exponentially. Additionally, the authors present an algorithm for the batch version of the problem, where we are given multiple patterns at once and want all substrings matching at least one of the given patterns. The algorithm for this problem constructs a trie of the query patterns and matches $S$ against the trie in a linear sweep.

Deng, Wang, and Liu [19] develop a string-based approach. Their algorithm constructs a suffix tree of the text and then matches a blocked pattern by iteratively generating all possible strings matching a mass of the pattern and searching them in the suffix tree. While this approach is memory-efficient, the worst case running time is linearly depending on the length of the text $n$.  

111
4.2 Problem Definition

Given a string $S = a_1 \ldots a_n$, two substrings $S_{i,j}$ and $S_{j+1,l}$ with $1 \leq i \leq j < l \leq n$ are consecutive in $S$. A pattern is a sequence $P = (p_1, \ldots, p_m)$ of non-zero masses $p_i \in \mathbb{N}$. The pattern $P$ matches a string $S = a_1 \ldots a_n$ with $n \leq m$ if $S$ can be partitioned into $m$ consecutive, non-empty substrings $S_1, \ldots, S_m$ such that $m(S_i) = p_i$ for all $i = 1, \ldots, k$.

For example, the string $S = $ AGTADAG can be partitioned into two consecutive substrings AGT and ADAG. The substrings AG, TAD, and G are not a partition of $S$, as the penultimate character of $S$ is neither part of the second, nor of the third substring. The three substrings AG, TAD, A are not a partition of $S$, but of $S_{1,6}$. The pattern $(m(\text{GTA}), m(\text{DAAG}))$ matches $S$, as the first mass of the pattern is equal to $m(S_{1,3})$ and the second mass of the pattern is equal to $m(S_{4,7})$, while the pattern $(m(\text{A}), m(\text{TAD}), m(\text{AGG}))$ does not.

Problem 5 (ExactBPM). Given a string $S$ over an alphabet $\Sigma$ and a pattern $P$, find all substrings of $S$ that match $P$.

Properties of typical instances We consider all masses to be integers. This is not a limitation, as masses can be multiplied by an appropriate factor before rounding to integers in order to account for the mass spectrometers accuracy. We do not account for measurement errors and only consider substrings that exactly match the given pattern. However, the presented algorithms can be extended to additionally account for these errors. Finally, mass spectrometers measure relatively short strings with up to 30 characters. Hence, we are only interested in finding substrings with a bounded number of characters $k = 30$ [70].
Modification-tolerant Blocked Pattern Matching

A post-translational modification is defined by a tuple \((a, \delta)\) for \(a \in \Sigma\) and \(\delta \in \mathbb{Z}\) with \(m(a) + \delta > 0\). If a character \(a\) is modified by a modification \((a, \delta)\), this is denoted by \(a_\delta\). The mass of \(a_\delta\) is defined as \(m(a_\delta) := m(a) + \delta\). For example, the string \(S = A_\delta \Gamma \Delta_\delta\) has two modified characters and \(m(S) = m(AGA) + \delta_1 + \delta_2\). The offset \(\delta\) is usually smaller than the mass of the lightest amino acid (57 Dalton), but sometimes modifications with up to 200 Dalton mass offset are considered [69].

Given a string \(S\) and a set of modifications \(M\), a modified string of \(S\) results from modifying an arbitrary (possibly empty) subset of characters of \(S\), each with one modification in \(M\). For example, given \(S = AGA\) and \(M = \{(A, \delta)\}\), the only four modified strings of \(S\) are \(A_\delta \Gamma \Delta_\delta\), \(A_\delta \Gamma \Delta_\delta\), \(A_\delta \Gamma \Delta_\delta\), and \(S\) itself.

**Problem 6 (MODBPM).** *Given a string \(S\) over an alphabet \(\Sigma\), a set of modifications \(M\), and a pattern \(P\), find all substrings \(S'\) of \(S\) such that a modified string of \(S'\) matches \(P\).*

Note that this problem is different from the approximate BPM problem defined in [69]. The problem defined in [69] does not consider a fixed set of predefined modifications \(M\), but rather considers a single, but unexpected modification (“blind” search).

Mutation-tolerant Blocked Pattern Matching

Given a string \(S = a_1 a_2 \ldots a_n\), we consider three types of mutations:

- **insertion** of \(b \in \Sigma\) at position \(i\):
  \[ S^+(i, b) := a_1 \ldots a_{i-1} b a_i \ldots a_n \]
• substitution of the $i$-th character with $b \in \Sigma$:
  \[ S^i(i, b) := a_1 \ldots a_{i-1} b a_{i+1} \ldots a_n \]

• deletion of the $i$-th character:
  \[ S^-(i) := a_1 \ldots a_{i-1} a_{i+1} \ldots a_n \]

A mutated string of $S$ is a string that results from applying a single mutation (insertion, substitution, deletion) to a single character of $S$. For example, three mutated strings of a string $S = AGTAG$ are $S^+(3, C) = AGCTAG$, $S^-(2) = ATAG$, and $S^i(4, G) = AGTGG$, while $AGATG$ is not a mutated string of $S$.

**Problem 7 (MutBPM).** *Given a string $S$ over an alphabet $\Sigma$ and a pattern $P$, find all substrings $S'$ of $S$, such that a mutated string of $S'$ matches $P$.*

In proteomics, limiting the number of possible mutations to a single mutation is reasonable, because mutations rarely occur in close proximity [70].

### 4.3 Exact Blocked Pattern Matching

We can now present our algorithm MS-BPM for solving ExactBPM. Our aim is to construct a memory-efficient string-based index data structure that does not require an explicit translation of a pattern into all matching strings. As mentioned above, we only consider substrings of a given string $S$ up to a constant length $k$. MS-BPM constructs a trie of all substrings of $S$ of length at most $k$. In addition, it computes a collection of priority search trees, one for each distinct mass of the substrings of $S$ up to length $k$. The priority search tree for mass $m$ stores all trie vertices that represent substrings with mass $m$. 
The trie vertices are stored as two-dimensional points in the plane based on the preorder and postorder traversals. In the query phase, MS-BPM uses the priority search trees to find all trie vertices of a given mass that are in the subtree of a given trie vertex. The algorithm only visits trie vertices that represent substrings that match a prefix of the given pattern.

**Preprocessing**  Given a string $S$, MS-BPM stores all substrings of $S$ with length $k$ and all suffixes with length at most $k$ in a trie data structure. A trie (keyword tree, [39]) is a rooted directed tree, where each edge is labeled by exactly one character. The outgoing edges of a vertex have distinct labels. Every stored string is represented by the concatenation of the edge labels of exactly one path from the root to some vertex in the trie. A vertex $v$ represents a string $T$ if the concatenation of the edge labels on the path from the root to $v$ is equal to $T$. Moreover, a vertex representing a string $T$ has mass $m(T)$.

Assume that the query pattern consists of a single mass $p_1$. We can find all matching substrings of $S$ in the trie by visiting all vertices with mass $p_1$. This can be done efficiently by concatenating all vertices with equal mass in linked lists. We keep an array of pointers for each distinct mass pointing to some vertex with that mass. In the query phase, it remains to check, if there is a pointer for mass $p_1$, and, if such a pointer exist, traverse all vertices in the corresponding linked list.

However, there is no obvious generalization of this approach for query patterns $P = (p_1, \ldots, p_m)$ with multiple masses. Suppose that we consider the masses from $p_1$ to $p_m$. We denote the partial sum of the first $i$ masses in the pattern by $P_i := \sum_{j=1}^{i} p_j$. We do not want to visit all vertices with mass $P_i$ for $i > 1$, but only in those that
Figure 4.1: (left) Trie storing the substrings of $S = \text{AAGAGAGGA}$ up to length $k = 3$. The preorder and postorder traversal ranks are written next to the vertices. (right) Geometric representation of the trie based on the preorder and postorder traversal ranks. Every vertex is represented by a point with $x$-coordinate equal to the preorder rank and $y$-coordinate equal to the postorder rank. Ancestors of a vertex are in the north-west quadrant of the vertex, descendants in the south-east quadrant.

represent a string that matches the $i$-th prefix pattern $(p_1, \ldots, p_i)$. In other words, we want to visit only those vertices with mass $P_i$ that are descendants of some vertex with mass $P_{i-1}$. A linked list does not store the necessary structural information for such a traversal. Instead, we make use of a geometric representation of the trie based on the preorder and postorder traversal ranks (Figure 4.1) to retrieve the necessary structural information.

We denote the preorder traversal rank of a vertex $v$ by $\text{pre}(v)$
and the postorder traversal rank by $post(v)$. We embed the trie in the plane by placing each vertex $v$ at coordinate $(pre(v), post(v))$. We can easily determine the ancestor relationship \cite{20} of two vertices $v$ and $w$ using their coordinates:

**Proposition 1** \cite{20}. A vertex $v$ is an ancestor of a vertex $w$ if and only if $pre(v) < pre(w)$ and $post(v) > post(w)$.

Similar numbering schemes have been proposed \cite{54} that can also be updated more efficiently if the trie is modified.

Considering the embedding of the trie in the plane (Figure 4.1), we note that all descendants of a vertex $v$ are in the south-east quadrant of $(pre(v), post(v))$. We use priority search trees to efficiently retrieve the descendants of a given vertex with a specific mass. A priority search tree \cite{64} is a data structure that stores a dynamic collection of two-dimensional points, i.e. ordered pairs $(x, y)$ of positive, upper-bounded integers $0 \leq x, y \leq k$. Besides inserting and deleting points, the following operations are supported: Given integers $x_1, x_2$, and $y_1$, (a) enumerate all points $(x, y)$ with $x_1 \leq x \leq x_2$ and $y \leq y_1$ and (b) find a point $(x, y)$ with minimal $x$-coordinate among all points with $x_1 \leq x \leq x_2$ and $y \leq y_1$. The time complexity of both operations is in $O(\log(n) + k)$, where $n$ is the number of stored points and $k$ is the number of reported points. We construct one priority search tree for each distinct mass of the vertices in the trie. The priority search tree for mass $m$ stores all trie vertices with mass $m$ (Figure 4.2). Then, we can efficiently retrieve all vertices with mass $P_i$ that are descendants of a given vertex with mass $P_{i-1}$.

**Pattern query** MS-BPM queries a pattern $P = (p_1, \ldots, p_m)$ by iteratively considering the pattern masses from $p_1$ to $p_m$. When con-
considering the pattern mass \( p_i \), the algorithm visits the trie vertices with mass \( P_i \) that represent a string that matches the prefix pattern \((p_1, \ldots, p_i)\). In the first step, MS-BPM visit vertices with mass \( P_1 \). When considering the next mass \( p_2 \), the algorithm only visits vertices with mass \( P_2 \) that are descendants of a vertex with mass \( P_1 \). In general, the algorithm visits the descendants with mass \( P_{i+1} \) of each previously visited vertex with mass \( P_i \). In this way, the algorithm only visits a vertex \( v \) with mass \( P_i \) if there is a vertex with mass \( P_j \) for every \( j < i \) on the path from the root to \( v \). Hence, \( v \) represents a string that matches \((p_1, \ldots, p_i)\).

We consider two options for traversing the trie in this way. We can either traverse the trie in a breadth-first search or in a depth-first search. In the breadth-first search, the algorithm first adds all vertices
with mass $P_1$ to a queue and then proceeds as follows until the queue is empty. It removes the next vertex $v$ with mass $P_i$ from the queue and queries the priority search tree storing the vertices with mass $P_{i+1}$ for all vertices with an $x$-coordinate in the interval $[pre(v), n_t]$, where $n_t$ is the number of vertices in the trie, and a $y$-coordinate smaller than $post(v)$. Then, it adds all reported vertices to the queue and continues with the next vertex. A disadvantage of this strategy is that the queue can quickly grow to a size linear in the number of trie vertices in the worst case.

Following the depth-first search strategy, the algorithm always stores at most $m$ vertices of the currently visited path (Algorithm 5 and Figure 4.3). The algorithm works on a search path that initially only contains the root vertex with mass $P_0 = 0$. Let $i$ be the length of the search path, $v$ be the last vertex (with mass $P_{i-1}$), and $v'$ the penultimate vertex (with mass $P_{i-2}$) of the path.

The function `exploreDown` queries the priority search tree of mass $P_i$ for the left-most descendant of vertex $v$ and expands the search path by the reported vertex if such a vertex exists. The function returns `true` if the current search path has been expanded and otherwise does not alter the search path returning `false`. The function `exploreRight` replaces $v$ by the next descendant of $v'$ with mass $P_{i-1}$ and the minimal preorder rank higher than $pre(v)$. To find this vertex, the function computes the left most vertex of mass $P_{i-1}$ with $x$-coordinate in the interval $[pre(v) + 1, n_t]$ and $y$-coordinate at most $\min(n_t, post(v'))$, where $n_t$ is the number of trie vertices.

While the breadth-first search requires one priority search tree operation for every visited vertex, the depth-first search requires two operations for each visited vertex. We implement depth-first search to reduce the memory-requirements in the query phase.
Algorithm 5: MS-BPM – Pattern Query

**Input:** pattern

```plaintext
currentPath ← ((1, n_t))  // stack for search path
expand ← true  // true if path gets expanded
```

```plaintext
do
  if currentPath.size() == pattern.size() + 1 then
    report currentPath
    currentPath.pop()
    expand ← false
  else
    if currentPath.size() == pattern.size() + 1 then
      expand ← exploreDown(currentPath,pattern)
    else
      expand ← exploreRight(currentPath,pattern)
  while currentPath not empty
```

**Time and Space Complexity**

First, we analyze the time complexity of our algorithm. Let $n$ be the length of $S$ and $k$ be an upper bound on the length of a matching substring in $S$. As mentioned above, we consider $k$ to be a constant, e.g. $k = 30$ [70].

**Preprocessing** We build the trie storing $n$ substrings of $S$ in $O(n)$ time. The trie contains up to $nk \in O(n)$ vertices, as every substring is represented by a path of constant length $k$. We can compute $pre(v), post(v)$, and the mass of $v$ for all vertices $v$ in $O(n)$ time. In
Figure 4.3: Matching the query pattern $P = (m(AG), m(A))$ in $S = AAGAGAGGA$ (Figure 4.2). The vertices stored in a trie are filled, the reported vertex is denoted by a filled square. (I) Query for leftmost vertex $v$ with mass $m(AG)$. (II) Query for leftmost descendant of $v$ with mass $m(AGA)$. (III) Query for next vertex $w$ with mass $m(AG)$. (IV) Query for descendant of $w$ with mass $m(AGA)$.

the construction of our index data structure, we use in-place min-max priority search trees [16]. Let $T_p$ be the number of trie vertices with mass $p$ and $T = \max_p T_p$. Each priority search tree can be constructed in $O(T \log(T))$ time and we construct at most $nk \in O(n)$ priority search trees in total. Hence, the preprocessing phase takes $O(n \cdot T \log(T))$ time.

**Pattern query** In the query phase, the algorithm visits a vertex of mass $P_i$ only if it is a descendant of a vertex with mass $P_{i-1}$. Therefore, every visited vertex of mass $P_i$ matches the prefix pattern $(p_1, \ldots, p_i)$. In the worst case, the algorithm visits $O(n)$ vertices. For every visited vertex, the algorithm performs two priority search tree operations. Therefore, the worst-case running time can be bounded by $O(n \cdot \log(T))$.

Let us analyze the average case query time complexity for a given
pattern and a random string $S$. The analysis is similar to the analysis of the suffix tree-based algorithm of Deng, Wang, and Liu [19]. Consider a string $R$ with length $l$ that matches some prefix $(p_1, \ldots, p_k)$ for some $1 \leq k \leq m$ of the given pattern $P$. There exists exactly one partition of $R$ into substrings $R_1, \ldots, R_k$ such that $m(R_i) = p_i$ for all $i = 1, \ldots, k$. Let $l_i$ be the length of the $i$-th substring $R_i$. We call $l_1, \ldots, l_k$ a partition of $R$.

How many strings with length $l$ that match the prefix $(p_1, \ldots, p_k)$ of the given pattern exist? Let $N_{p_i,j}$ be the number of strings with mass $p_i$ and length $j$. For a given partition $l_1, \ldots, l_k$, there exist $\prod_{i=1}^k N_{p_i,l_i}$ strings that match the prefix. We can express this product as a function of $l := \sum_{i=1}^k l_i$ by defining $r_{i,j} := (N_{p_i,j})^{1/j}$ and $r := \max_{i,j} r_{i,j}$. Then, $\prod_{i=1}^k N_{p_i,l_i} = \prod_{i=1}^k (r_{p_i,l_i})^{l_i} \leq \prod_{i=1}^k r^{l_i} = r^l$. That is, there exist at most $r^l$ substrings of length $l$ that match a prefix of the given pattern. Note that there exist $2^l$ partitions for a string of length $l$. Hence, the number of strings of length $l$ that match a prefix of $P$ is bounded by $(2r)^l$. Let us now consider a random string $S$ with uniform character frequency $1/|\Sigma|$. Deng et al. [19] show that the expected number of substrings that match a prefix of the given pattern is bounded if $1 < 2r \leq |\Sigma|$.

**Lemma 3 ([19]).** If $1 < 2r \leq |\Sigma|$, then the expected number of substrings of arbitrary length matching a prefix of a given pattern $P$ in random string $S$ over $\Sigma$ with uniform character frequency is in $O(n \log |\Sigma| 2r)$.

Our algorithm visits exactly one vertex for every substring of $S$ that matches a prefix of the given query pattern. For every visited vertex, the algorithm queries two priority search trees (exploreDown and exploreRight). Therefore, the average case time complexity of
our algorithm is in $O(n^{\log_2 2r} \cdot \log(T) + M)$, where $M$ is the number of substrings of $S$ that match the pattern $P$. For practical applications, Deng et al. [19] estimate $r \approx 5.7$, which leads to an average case running time sublinear in the size of the database.

**Space complexity** In the preprocessing phase, we construct a trie with at most $nk$ vertices using $O(n)$ space. Moreover, we construct up to $nk$ priority search trees. A priority search tree storing $T$ vertices requires $O(T)$ space [16]. As each vertex is stored in exactly one priority search tree, the total space complexity of the index data structure is in $O(n)$. In the query phase, our algorithm visits the vertices following a depth-first strategy and keeps at most $m$ vertices of the current search path in memory. Thus, the additional memory required in the query phase is in $O(m)$ if we assume that the matching substrings are not stored, but reported immediately.

### 4.4 Approximate Blocked Pattern Matching

#### 4.4.1 Modification-tolerant BPM

In this section, we present the algorithm MS-BPM-mod for the modification-tolerant BPM problem. The algorithm is based on the algorithm MS-BPM and extends the index data structure to account for a predefined set of modifications. We first explain the main idea in an example and then describe the complete algorithm in general.

**Example** Consider a binary alphabet $\Sigma = \{A, B\}$ with $m(A) = 2$ and $m(B) = 6$ and a set of modifications $M = \{(A, 1), (B, -1)\}$. Assume that we are given a string $S = AAAB$ and a pattern $P = (5)$. Note
that there exists no unmodified string with mass $p_1 = 5$ and that we want to find all substrings $T$ of $S$ that have a modified string with mass $p_1$. Storing all possible modified strings of $S$ in the index data structure would require an extensive amount of memory. However, we can guess the mass $p'$ of the unmodified substring $T$ and find all substrings of $S$ with mass $p'$. For simplicity, we only consider the case, where at most one modification is applied to $T$. An unmodified substring $T$ either has mass 4 (and a modification at some character $A$ leading to an offset of +1) or mass 6 (with a modification of $B$).

In this example, there are two possible unmodified substrings $AA$ (with mass 4) and $B$ (with mass 6) that have modified strings with mass $p_1$. In general, there can be many possible unmodified strings and we do not want to check which of them is a substring of $S$. Therefore, we rather consider the distinct masses of these unmodified strings and search for substrings with these masses in $S$. Concretely, we search for all substrings of $S$ with mass 4 or 6. Unfortunately, not all substrings we find in this way can be modified to match $p_1$. For example, we find the substring $AAA$ with mass 6 that has no modified string with mass $p_1$.

The main idea of our algorithm is to build an index, such that we can decide in constant time if a substring of $S$ with a given mass has a modified string that matches a prefix of the given pattern. When we guess the mass $p'$ of the unmodified string $T$, we implicitly fix the modification that needs to be applied to $T$. Therefore, we need to check if this modification is applicable to the substrings of mass $p'$ in $S$, i.e. which of the substrings contains at least one character of the chosen modification. For example, we need to apply the modification $(B, -1)$ to a substring of $S$ with mass 6 in order to get a modified string with mass $p_1 = 5$. As this modification is not applicable to
the substring AAA, this substring is not matching the pattern.

Assume that AAA is represented by a vertex \( v \) in the trie. How can we check if the path from the root to \( v \) has an edge with label B, i.e. whether \((B, -1)\) is applicable? We can store the necessary information in the preprocessing step. We store for each vertex the last occurrence of character B on the path from the root to this vertex. In the query phase, we then only have to check the last occurrence of B when visiting \( v \). In general, if we visit a vertex matching the \( i \)-th prefix of the query pattern, we check if the character that needs to be modified is an edge label on the path starting at the predecessor of \( v \) matching the \((i - 1)\)-th prefix and ending in \( v \).

**Algorithm** The algorithm MS-BPM-mod computes the same index data structure consisting of a trie and a collection of priority search trees as MS-BPM. In addition, the algorithm stores for every trie vertex \( v \) and every character \( x \) with a modification in \( M \) the preorder rank \( \text{last}(v, x) \) of the source vertex of the last edge with label \( x \) on the path from the root to \( v \).

In the query phase, the algorithm is given a query pattern \( P = (p_1, \ldots, p_m) \). Similar to MS-BPM, the algorithm operates on a search path that initially only contains the root of the trie. Additionally, MS-BPM-mod stores the chosen modifications for each matched pattern mass. That is, if the search path matches the \( k \)-prefix of \( P \), the sequence of chosen modifications contains \( k \) vectors, each of size \(|M|\), indicating how often each modification is used for matching the corresponding pattern mass.

The modifications for a vertex are chosen in a predefined, but arbitrary order. We use the lexicographic order of the vectors describing the set of chosen modifications. In general, for a mass \( p_i \)
there are at most \( \lfloor p_i / (m(x) + \delta) \rfloor \) possible modifications \((x, \delta)\) in a substring that matches the mass \( p_i \). In our implementation, we consider at most one modification of each type for each pattern mass \( p_i \), as the pattern masses are usually small.

The algorithm uses the same traversal scheme as MS-BPM: We denote the last vertex of the search path by \( v \) and the penultimate vertex by \( v' \). First, MS-BPM-mod tries to expand the search path using the function \text{exploreDown}. If this is not possible, it executes \text{exploreRight} that tries to replace the vertex \( v \) with the next vertex in the subtree of \( v' \) (with respect to the preorder rank) that has the same mass as \( v \). However, in contrast to MS-BPM, the algorithm does not remove \( v \) from the search path if no such next vertex exists. Instead, MS-BPM-mod continues with considering the next set of modifications. In this way, MS-BPM-mod also considers vertices with mass \( v + \Delta \) for all offsets \( \Delta \) of possible combinations of modifications in the subtree of \( v' \). The size of the search path is reduced as soon as all modifications have been considered. Whenever the algorithm expands or modifies the search path, it first checks if the corresponding set of modifications is applicable for the string represented by the path from \( v' \) to \( v \). As mentioned above, this can be done efficiently by checking if \( \text{last}(v, x) \geq \text{pre}(v') \) for each character \( x \) that is chosen to be modified.

Assuming a constant-sized alphabet and a constant-sized set of considered modifications \( \mathcal{M} \), both the space and time complexity of MS-BPM-mod are equivalent to the complexities of MS-BPM. In the preprocessing phase, the algorithm stores at most \(|\Sigma|\) additional preorder ranks for each trie vertex. The query procedure does not consider \( m \) different masses for a pattern of length \( m \), but \( c \cdot m \), where \( c \) is the number of possible combinations of modifications.
The number of possible combinations is constant in practice, as the length of a matching substring is bounded by a constant.

4.4.2 Mutation-tolerant BPM

Mutation-tolerant blocked pattern matching is another variant of approximate BPM. We can use the algorithm MS-BPM-mod presented in the previous section to solve this problem as well, as mutations can be represented by modifications. For every character \( a \in \Sigma \), we can add the following modifications to \( M \):

- \((a, -m(a))\) representing the deletion of \( a \),
- \((a, m(b))\) for each \( b \in \Sigma \) representing the insertion of \( b \), and
- \((a, m(b) - m(a))\) for each \( b \in \Sigma \) representing the replacement of \( a \) by \( b \).

Moreover, MS-BPM-mod can easily be modified to allow for at most one modification when querying a pattern. A disadvantage of this approach is the large size of the set of modifications, as \(|M| \in \mathcal{O}(|\Sigma|^2)\).

In this section, we describe another algorithm called MS-BPM-mut that solves MutBPM. We will see that this algorithm can also be used for considering a unrestricted version of approximate BPM, where no set of possible modifications are given. Unrestricted approximate block pattern matching is of interest in practice to analyze unexpected modifications (known as blind modification search).

Assume that we are given a query pattern \( P = (p_1, \ldots, p_m) \) with a mutation leading to an offset at pattern mass \( p_i \). MS-BPM can find strings that match the prefix pattern \((p_1, \ldots, p_{i-1})\) and the suffix pattern \((p_{i+1}, \ldots, p_m)\). Let \( T \) be a string that matches the prefix pattern,
Figure 4.4: Trie for string ABAA with $k = 3$. Each vertex $v$ has pointers (dashed, gray) to the first vertex of each path representing the same string as $v$.

$T'$ be a string that matches the suffix pattern, and $v$ the trie vertex representing $T$. A combination of $T$ and $T'$ matches $P$ if

- $T'$ is represented by a path in the subtree of $v$ and if
- the difference between the mass $p_i$ and the mass of the first vertex $v'$ of the path representing $T'$ in the subtree of $v$ can be explained by a mutation in the string represented by the path from $v$ to $v'$.

A trivial check of the first condition requires traversing the whole subtree of $v$. Our algorithm MS-BPM-mut tries to check this condition without traversing the subtree by storing additional information in the index data structure. We store for each vertex $w$ in the trie a pointer to the first vertex of each path in the trie that represents the same string as $w$ (Figure 4.4). Vertices that represent short strings with few characters usually have many pointers, while the number of pointers rapidly decreases with increasing length of the represented string.
MS-BPM-mut considers every position $i$ of the mutation in the pattern from 1 to $m$. Then it computes all substrings matching the corresponding prefix and suffix pattern. Finally, the algorithm tries to combine each pair of substrings matching the prefix and the suffix pattern using the additional information of the index data structure. If $i$ is small, we expect many substrings matching the prefix pattern and few substring matching the suffix pattern. Analogously, if $i$ is close to $m$, we expect many substrings matching the suffix and few substrings matching the prefix pattern. These corner cases leave some space for optimization on the implementation level. For example, if $i$ is small, it might be faster to perform a local extension of the substrings matching the suffix pattern as proposed in [70].

Another improvement would be to check the pointers of the trie vertices already while searching the substrings matching the suffix pattern. If the vertex matching a suffix has a pointer in a subtree matching the prefix pattern, all its predecessors also have a pointer in the same subtree. Therefore, we only have to continue the search at vertices that have a pointer in a subtree of a vertex that represents a string that matches the prefix pattern. After finding all strings matching the prefix pattern, we can store the intervals of preorder ranks of the vertices in the subtrees of vertices matching the prefix pattern (Figure 4.5). While matching the suffix pattern, we only consider the descendants of a vertex if it has a pointer that points to a vertex with preorder rank in one of the intervals.
Figure 4.5: Improvement of suffix pattern queries. (left) Paths that are visited while matching the prefix pattern. The two vertices with mass $P_1$ have descendants with preorder ranks in the intervals depicted below the trie. (right) Intervals of preorder ranks of the subtrees of vertices visited while matching the prefix pattern. For example, we only have to visit vertices with a pointer to the interval for mass $p_4$ when matching the suffix pattern $(p_{m-5}, \ldots, p_m)$.

### 4.5 Experimental Evaluation

We implement the three proposed algorithms MS-BPM, MS-BPM-mod, and MS-BPM-mut in C++ and evaluate their performance by matching synthetic patterns and patterns generated from real spectra against sequence databases of various sizes. We analyze the running times of our algorithms with similar methods as used in [70, 19]. A comparison with the algorithms proposed by Ng et al. [70] and Deng et al. [19] was not possible, as the implementations are not publicly available. In [70], the authors only report running times of
their algorithm for the batched version of the problem, where the algorithm is given all query patterns at once instead of one by one. Additionally, we analyze the identification rates of MS-BPM-mod compared to MS-BPM on datasets measured with various types of instruments. All tests were performed on the ETH HPC-cluster Euler.

**Dataset** We considered the complete human proteome database from Swiss-Prot and a subset consisting of the first million characters in the human proteome database for analyzing the running times of our algorithms. For evaluating how many spectra remain identifiable after the filtration using MS-BPM and MS-BPM-mod, we considered the Standard Protein Mix Database [50], specifically the sequence database and Mix2-spectra measured with three types of instruments (LTQ, LCQ-Deca, and QTOF1). The query patterns were either generated directly from substrings in the database or from real spectra using MS-Gapped-Dictionary [44].

**Running time analysis** We analyzed the query times of our algorithms for synthetic patterns and patterns generated with MS-Gapped-Dictionary [44] from spectra of the Standard Protein Mix Database [50]. To analyze the running time of our algorithms depending on the structure of the query patterns, we generate synthetic patterns from sequences of the human proteome. Given a string $T = a_1 \ldots a_n$ a pattern with gap size $k$ is computed by partitioning $T$ into substrings of length $k$ and taking the mass of these substrings as pattern masses. That is, we generate the pattern of $T$ with gap size $k$ is $(m(a_1 \ldots a_k), m(a_{k+1} \ldots a_{2k}), \ldots, m(a_{\lceil n/k \rceil \cdot k+1} \ldots a_n))$.

In this evaluation, we consider one million substrings with 20 characters of the human proteome and generate patterns for gap sizes
\[k = 4 \text{ and } k = 6.\] We use a mass resolution of 0.01 Dalton, i.e. we multiply all masses with factor 100 before rounding them to integers. The reported query times do not include the time for constructing the index data structure (963 seconds for MS-BPM, 1322 seconds for MS-BPM-mod with one modification, 2615 seconds for MS-BPM-mod with two modifications), as this step is independent of the query patterns and has to be done only once for a given string \(S\) and the set of considered modifications \(M\).

MS-BPM matched one million patterns with gap size \(k = 4\) against the complete human proteome in 303 seconds using 10 GB memory. The average query time increases with increasing gap size. This is due to the fact that with increasing gap size, typically more descendants of a visited vertex need to be checked. MS-BPM takes 2410 seconds for matching one million patterns with gap size \(k = 6\).

Deng et al. [19] similarly observe significantly increased query times for increasing gap sizes in the analysis of their algorithm.

We analyzed the running time of MS-BPM-mod for one and two modifications on the set of synthetic patterns with \(k = 4\). First, we run the algorithm considering Methionine oxidation (+15.99 da) as possible modification. MS-BPM-mod took 2591 seconds seconds for matching one million patterns with gap size \(k = 4\). Then, we additionally considered Cysteine carbamidomethylation (+57.02 da). Considering both modifications, the query time for one million patterns was 4470 seconds.

We analyzed the running times of our algorithms on patterns generated from real spectra. We considered 30000 spectra from the Standard Protein Mix Database, Mix 2, QTOF1 [50] and used MS-Gapped-Dictionary [44] (Version 102010) with default parameters for generating the patterns. MS-Gapped-Dictionary generates pat-
terns for integer masses. Therefore, we used mass resolution 1 da when matching the patterns. Similar to the evaluation in [70], we considered only patterns, where no mass exceeds 500 Dalton, and removed duplicates before matching the patterns. Finally, we matched the resulting 116,000 patterns against the first one million characters of the human proteome. Again, the reported query times do not include the time for constructing the index data structure (93 seconds for MS-BPM, 124 seconds for MS-BPM-mod with one modification, 145 seconds for MS-BPM-mod with two modifications, and 571 seconds for MS-BPM-mut).

MS-BPM takes 19.5 seconds for matching all patterns against the database using less than 1 GB memory. When considering only Methionine oxidation, MS-BPM-mod takes 122 seconds. Considering two possible modifications (Methionine oxidation and Cysteine carbamidomethylation), MS-BPM-mod matches all patterns in 432 seconds. MS-BPM-mut has significantly larger query times: 10040 seconds for the considered spectra.

Rate of identifiable spectra We evaluated, how many spectra can be correctly annotated using the patterns generated from MS-Gapped-Dictionary when considering exact, respectively approximate BPM. A spectrum can only be annotated with the correct sequence if at least one of the patterns that are derived from this spectrum matches the correct sequence. Otherwise, the correct sequence will get discarded during the filtering procedure and, hence, not be considered in the later steps of the identification process.

We analyze spectra of three different instruments (LTQ, LCQ-Deca, and QTOF) from the Standard Protein Mix Database [50] and generated query patterns using MS-Gapped-Dictionary with default
Figure 4.6: Identifiable spectra when using MS-BPM and MS-BPM-mod.

parameters. We considered the top-scoring sequences of a database search as provided in [50] as ground truth. We call a spectrum identifiable if the true sequence is not discarded by the filtration using exact, respectively approximate BPM.

The available version of MS-Gapped-Dictionary does not support generating patterns for modified or mutated peptides. Therefore, spectra of modified or mutated peptides might only be identifiable using approximate BPM for filtration. We filtered the database using these patterns with our algorithms MS-BPM and MS-BPM-mod and compared the number of identifiable spectra for both algo-
rithms. For modification-tolerant BPM, we considered Methionine oxidation and, in a second experiment, additionally Cysteine carbamidomethylation.

The number of identifiable spectra for all three types of instruments is shown in Figure 4.6. The figure shows that the number of identifiable spectra increase for all three instrument types by at least 17% when considering post-translational modifications.
Chapter 5

Concluding Remarks

In this thesis, we studied algorithmic problems in mass spectrometry-based proteomics. Specifically, we considered the peptide sequencing problem, i.e. the problem of identifying the amino acid sequence of one or multiple peptides in a sample from data obtained in a mass spectrometry experiment. In such an experiment, multiple copies of a peptide are fragmented – each copy at a single, random position – into prefix and suffix fragments and a mass spectrum of these fragments is acquired. Given the mass of the peptide and the pattern of fragment masses, the amino acid sequence of the peptide has to be reconstructed.

While some approaches for peptide sequencing try to reconstruct the sequence of a peptide from scratch (de novo) by interpreting the fragment masses and their pairwise differences, other approaches query databases of known sequences and search for the sequence with
the most similar expected fragmentation pattern. There exist several approaches that combine database search with de novo sequencing. In such approaches, de novo sequencing is usually applied first and the result of this step is then used to speed up the database search. Besides speeding up the database search by de novo techniques, it might also be interesting to study approaches that use the database to speed up de novo sequencing. The aim here would be to reconstruct known sequences fast, while still be able to identify unknown peptides. Such approaches could use the database as a heuristic to guide the search for the optimal solution (cf. the well-known $A^*$ algorithm).

Whether we consider de novo or database search algorithms, in both cases we need to define a notion of similarity to compare an amino acid sequence and a fragmentation pattern. Usually, such a notion is based on a model of peptide fragmentation that defines the expected fragment masses (theoretical spectrum) of a given amino acid sequence. The similarity between a sequence and a fragmentation pattern is then defined by comparing the theoretical spectrum of the sequence with the given fragmentation pattern.

In Section 3.2, we propose a new scoring model for comparing a sequence with a fragmentation pattern, namely to minimize the size of the symmetric difference between expected and measured fragment masses. We study the de novo peptide sequencing problem with this scoring model and present an efficient algorithm. Besides de novo peptide sequencing, this scoring model is also applicable in database search. Moreover, the scoring model is also worth studying in the context spectral alignments [76, 6], i.e. the problem of comparing fragmentation patterns of similar peptides and aligning their fragment masses.

A challenging problem in peptide sequencing is the presence of
mixture spectra. Such spectra result from the simultaneous analysis of multiple peptides, whose fragment masses cannot be easily distinguished from the measurement data. In this case, we face the additional problem of first assigning each mass to one or multiple peptides and then to reconstruct their amino acid sequence. We study de novo sequencing for multiple peptides in Section 3.4. However, the complexity of de novo sequencing for multiple peptides is still unknown and theoretical insights for this problem might also lead to better algorithms for de novo peptide sequencing.

**List of Contributions**

The results presented in this thesis were developed throughout my doctoral studies in collaboration with many others, who I enjoyed to work with.

The study on the symmetric difference scoring model in Section 3.2 has been published in [35, 90] and was developed in collaboration with Simon Rösch during his master thesis supervised by Peter Widmayer, who contributed in many fruitful discussions do the algorithmic design. Ludovic Gillet provided and prepared the data for the experimental evaluation and contributed to the design of the experimental evaluation. Simon Rösch implemented most parts the algorithm.

The research on exploiting the liquid chromatographic information in de novo peptide sequencing was motivated by a brief initial discussion with Witold Wolski and Christian Panse. The algorithms presented in Section 3.3 have been published in [32] and were joint work with Tomas Hruz, Yves Frank and Valentin Venzin during the bachelor theses of Yves Frank and Valentin Venzin. The algorithm
for the position-dependent prediction model was developed with Valentin Venzin, while the neighborhood-based prediction model was studied with Yves Frank. Tomas Hruz and Peter Widmayer supervised the work and contributed in many helpful discussions. The initial implementation of the algorithms is the work of Yves Frank and Valentin Venzin.

De novo sequencing from mixture spectra (Section 3.4, [91]) is joint work with Tomas Hruz, Hannes Röst, and Ludovic Gillet. Hannes Röst and Ludovic Gillet patiently answered all our questions on the instrumentation, the experimental design, and on characteristics of the data. The development of the algorithm is joint work with Tomas Hruz. Hannes Röst and Ludovic Gillet contributed to the experimental evaluation of the algorithm, provided the data, and performed preparatory parts of the experiments.

The study on blocked pattern matching (Chapter 4, [89]) was motivated by initial discussions with Peter Widmayer, who also gave feedback during the development of the algorithms.

The authors of the mentioned publications and manuscripts, and especially Peter Widmayer and Tomas Hruz, gave many valuable feedback on the writing and the presentation of the results.
Bibliography


[54] Quanzhong Li and Bongki Moon. “Indexing and Querying XML Data for Regular Path Expressions”. In: *Proceedings of*


152


[82] Olga T. Schubert, Ludovic C. Gillet, Ben C. Collins, Pedro Navarro, George Rosenberger, Witold E. Wolski, Henry Lam,


