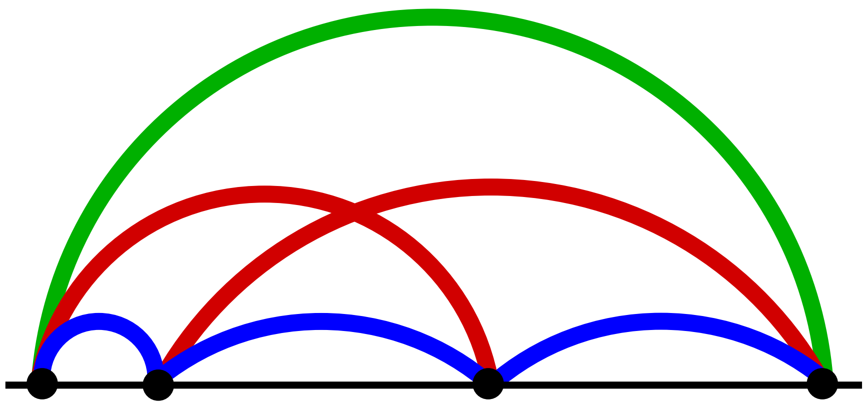
Diss. ETH No. 15258 2003

Algorithms and Hardness Results for DNA Physical Mapping, Protein Identification, and Related Combinatorial Problems



Mark Cieliebak

Diss. ETH No. 15258, 2003

Algorithms and Hardness Results for DNA Physical Mapping, Protein Identification, and Related Combinatorial Problems

A dissertation submitted to the
Swiss Federal Institute of Technology, ETH Zurich
for the degree of Doctor of Technical Sciences

presented by
Mark Cieliebak
Dipl. Informatiker, University of Dortmund
born February 28, 1970 in Hagen, Germany

accepted on the recommendation of
Prof. Dr. Peter Widmayer, examiner
Prof. Dr. Thomas Erlebach, co-examiner

Abstract

In this thesis, we focus on two applications of digestion experiments, namely *physical mapping of DNA* and *protein identification*, and study the computational complexity of combinatorial problems that arise in this context.

Digestion experiments play an important role in molecular biology. In such experiments, enzymes are used to cleave DNA molecules or proteins at specific sequence patterns, the *restriction sites.* The resulting fragments are used in many different ways to study the structure of DNA and proteins, respectively.

In the Double Digest problem, we are given the lengths of DNA fragments arising from digestion experiments with two enzymes, and we want to find a physical map of the DNA, i.e., the positions of the restriction sites of the enzymes along the DNA sequence. Double Digest is known to be NP-hard. We show that the problem is even strongly NP-hard, even if the two enzymes always cut at disjoint restriction sites. Moreover, we show that for partial cleavage errors the problem to find solutions with a minimum number of errors is hard to approximate.

In the Partial Digest problem, we are given DNA fragment lengths arising from digestion experiments with only one enzyme, and we again ask for a physical map of the DNA. Neither a proof of NP-hardness nor a polynomial–time algorithm is known for Partial Digest. We study variations of Partial Digest that model missing fragments, additional fragments, and erroneous fragment lengths, and show that these variations are NP-hard, hard to approximate, and strongly NP-hard, respectively.

The Equal Sum Subsets problem, where we are given a set of positive integers and we ask for two subsets such that their elements add up to the same total, is known to be NP-hard. Equal Sum Subsets can be used to prove NP-hardness for Partial Digest variants. Motivated by this, we study variations of Equal Sum Subsets, where we, for instance, allow any positive rational factor between the sums of the two subsets. We give (pseudo–)polynomial algorithms or (strong) NP-hardness proofs, respectively, for several natural variations of EQUAL SUM SUBSETS.

In the second part of this thesis, we address the problem of protein identification. The *mass fingerprint* of a protein contains the masses of fragments that emerge when digesting the protein. Mass fingerprints are used, for instance, to search for proteins in large protein databases, without sequencing them. The MASS FINDING problem arises in this context. Here, we are given a mass M and a protein sequence, and we ask whether there is a fragment of the protein that has mass M. MASS FINDING can be solved easily in time linear in n, the length of the protein sequence. We present an algorithm that solves the problem even in sublinear time $O(\frac{n}{\log n})$. This algorithm uses a data structure that is generated in a preprocessing step, and that requires only linear storage space.

A different approach to identifying a protein is to establish its amino acid sequence (*de novo sequencing*). Here, a fragment of the protein (*peptide*) is dissociated, and the masses of the resulting pieces are measured using tandem mass spectrometry. This yields an *MS/MS spectrum* of the peptide. For the case of error–free data, algorithms exist that construct the amino acid sequence of a peptide from its MS/MS spectrum. We have implemented a software tool (Audens) that allows for de novo peptide sequencing even in the case of erronoeous data, and evaluated its performance on real–life spectra.

One problem that arises in the context of Audens is the DECOMPOSITION problem, where we ask whether a given mass can be represented as a sum of amino acid masses. This problem is known to be NP-hard. We show that DECOMPOSITION can be solved in polynomial time if the number of different amino acid masses is constant, or if the masses of all but a constant number of amino acids are polynomially bounded. On the other hand, we show that if we ask for the *minimum* or *maximum* number of amino acids whose masses add up to the given mass, then no polynomial–time algorithm can guarantee any constant approximation ratio (unless $\mathsf{P} = \mathsf{NP}$).

Zusammenfassung

In dieser Arbeit betrachten wir zwei Anwendungen von Verdau-Experimenten (*digestion experiments*): *physikalische Kartierung von DNS–Molekülen* und Identifikation von Proteinen. Wir untersuchen die algorithmische Komplexität von verschiedenen kombinatorischen Problemen, die in diesem Zusammenhang auftreten.

Verdau-Experimente spielen eine wichtige Rolle in der Molekularbiologie. In diesen Experimenten werden Enzyme verwendet, um DNS–Moleküle oder Proteine an bestimmten Sequenzmustern, den *Restriktionsmustern*, aufzuspalten. Die entstehenden Fragmente werden verwendet, um die Struktur der DNS-Moleküle bzw. Proteine zu untersuchen.

Beim Double Digest Problem sind die Längen von DNS–Fragmenten aus Verdau-Experimenten mit zwei Enzymen gegeben. Hieraus soll eine physikalische Karte der DNS berechnet werden, die die Positionen in der DNS–Sequenz angibt, an der die Restriktionsmuster der Enzyme auftreten. Das Double Digest Problem ist NP-schwer. Wir zeigen, dass es sogar stark NP-schwer ist, selbst wenn die beiden Enzyme die DNS stets an verschiedenen Positionen aufspalten. Ausserdem zeigen wir, dass Double Digest schwer zu approximieren ist, wenn ein Enzym die DNS an einer Position möglicherweise nicht spaltet, obwohl dort das Restriktionsmuster des Enzyms vorliegt (*partial cleavage error*).

Beim Partial Digest Problem sind Fragment-Längen aus Verdau-Experimenten mit nur einem Enzym gegeben, und wie bei Double Digest soll eine physikalische Karte der DNS berechnet werden. Es ist nicht bekannt, ob Partial Digest polynomiell lösbar oder NP-schwer ist. Wir untersuchen das Problem für die Fälle, dass einige Fragment-Längen in der Eingabe fehlen oder dass zusätzliche Längen vorkommen oder dass die Längen nicht exakt gemessen wurden. Wir zeigen, dass die entsprechenden Varianten von Partial Digest NP-schwer bzw. schwer zu approximieren bzw. stark NP-schwer sind.

Das Equal Sum Subsets Problem, bei dem n natürliche Zahlen gegeben sind und wir nach zwei Teilmengen suchen, deren Elemente sich zur selben Summe aufaddieren, tritt im Zusammenhang mit Partial Digest auf. Equal Sum Subsets ist NP-schwer. Wir untersuchen verschiedene Varianten von Equal Sum Subsets, z.B. wenn ein beliebiger positiver rationaler Faktor zwischen den Summen der beiden Teilmengen erlaubt ist, und geben (pseudo–)polynomielle Algorithmen an oder beweisen, dass sie (stark) NP-schwer sind.

Im zweiten Teil dieser Arbeit beschäftigen wir uns mit der Identifikation von Proteinen. Der *Fingerabdruck* eines Proteins enthält die Massen von Protein-Fragmenten, die beim Verdauen des Proteins entstehen. Fingerabdrücke werden z.B. verwendet, um ein Protein in einer Proteindatenbank zu suchen. In diesem Zusammenhang tritt das Mass Finding Problem auf, bei dem eine Masse M und eine Proteinsequenz gegeben sind und entschieden werden soll, ob das Protein ein Fragment der Masse M enthält. Das Mass Finding Problem kann in Zeit linear in n, der Länge der Proteinsequenz, gelöst werden. Wir präsentieren einen Algorithmus mit sublinearer Laufzeit $O(\frac{n}{\log n})$. Dieser Algorithmus verwendet eine Datenstruktur, die vorab berechnet wird und die nur linearen Speicherplatz benötigt.

Proteine können auch identifiziert werden, indem man ihre Aminosäuren-Sequenz bestimmt (*de novo sequencing*). Eine Methode hierfür spaltet zunächst das Protein in Fragmente (*Peptide*) auf. Die Peptide werden dann einzeln weiter zerkleinert, und die Massen der entstehenden Teilstücke werden mittels Massenspektrometrie bestimmt. Dies liefert ein *Tandem–Massenspektrum (MS/MS Spektrum)* für jedes einzelne Peptid. Es existieren effiziente Algorithmen, die aus einem MS/MS Spektrum die Aminosäuren-Sequenz des Peptids berechnen, falls die Daten fehlerfrei sind. Da diese Annahme jedoch i.d.R. auf reale Spektren nicht zutrifft, haben wir ein Sequenzierungs-Programm (Audens) implementiert, das auch Fehler in den Daten zulässt, und seine Qualität anhand von realen Spektren evaluiert.

Ein Problem, das im Zusammenhang mit Audens auftaucht, ist das Decomposition Problem, bei dem entschieden werden soll, ob eine gegebene Zahl sich als Summe von Aminosäuren-Massen darstellen lässt. Dieses Problem ist NP-schwer. Wir zeigen, dass das Problem in polynomieller Zeit lösbar ist, wenn die Anzahl der verschiedenen Aminosäuren-Massen konstant ist oder wenn es nur konstant viele Aminosäuren gibt, deren Masse nicht polynomiell beschränkt ist. Ausserdem betrachten wir die beiden Optimierungsvarianten, bei denen wir nach einer maximalen bzw. minimalen Anzahl von Aminosäuren fragen, deren Massen sich zu einer bestimmten Zahl aufsummieren. Wir zeigen, dass kein polymieller Algorithmus für diese beiden Optimierungsprobleme existiert, der einen konstanten Approximationsfaktor garaniert (falls $\mathsf{P} \neq \mathsf{NP}$).

Acknowledgements

In the past 4.5 years here in Zürich, I did not only write this thesis, but I also had the pleasure to meet and share time with many many great people. I believe that their friendship and support was the basis for every single line of this thesis, and I would like to express my sincere gratitude to:

Peter Widmayer, my supervisor, for his advice and patience, and for allowing me to do it my way;

Thomas Erlebach, for being my co–examiner, and for checking so many proofs in detail;

the members of our research group, for hundreds of Gipfeli and cakes, for helping me with complex hard– and software problems (such as writing a CD), for all the discussions on computer science, movies, and politics, and for proof–reading 27 versions of this thesis;

my co–authors, for all the fruitful discussions we had at the white and black board;

Giuseppe/Nicola/Paola/Peter, Jens, Sacha/Torsten, and Dirk/HaJo, for pointing me to these beautiful research problems: gathering autonomous robots, protein identification, de novo peptide sequencing, and the partial digest problem;

my WGs in Oerlikon, Wipkingen, and Küsnacht, for gas–oven heated kitchens, balcony potatoes, winter gardens with upside–down candles, and a mattress under the rainbow;

all my friends in Zürich, for distracting me not too much with climbing, (beach-)volleyball, running, Boogie–Woogie, movie nights, hiking, brötle, and skiing;

my biologists Angi, Anna, Danny, and Mark, for raising my interest in the beauty of nature, and Sacha and Franz for explaining it to me;

the Swiss doctors and hospitals, for successfully fixing Aussenbandriss, Gehirnerschütterung, Hexenschuss, and Kreuzbandriss;

meine Volleyball–Damen, die ich so gern durch die Halle jage und bei deren Spielen ich alle Höhen und Tiefen der Gefühlswelt erleben durfte;

Angi, para observar pájaros a las cinco en la mañana, para invitarme a Ecuador, para cambiar mi imagen del mundo feminino, y para ser una amiga extraordinaria;

Alexx, for an exciting Java course, and for many many discussions on life, the universe, and everything;

Benita, wo mini beschti Fründin und mini grossi Liebi isch, wil sie sich noed bschwere tuet, au wenn ich zmitts i dNacht ufschta um aechli ae geniali Idee uf zschriebe – wo sich denn fascht immer am nächschti Morge als völlige Seich useschtelle tuet;

Carsten, Jan, and Tordis, who managed to be close friends over more than 600km and 4 years;

Kai, for convincing me to come to Zurich, for proving we are not made for business, and for being there whenever I really need help and advice;

Katrin, for her power and spirit on the beach court, for cheering up my lunch times, and for her deep insights into men's soul;

Mark2, for climbing Cotopaxi and Züriberg with me, for the wonderful queensize bed, and for all the silent moments during our hikes;

Roland, for pushing the SOI to the limits, for exciting travels to three continents, and for providing striking insights into the internet, Islam, and loaded dice;

Snut and Hinrich, die ständig für Grundnahrungsmittel wie Kekse und Zitronentee gesorgt haben, bei denen ich je nach Laune Wände einreissen oder vor'm Kamin schlafen konnte, und die als Eltern einfach einzigartig sind;

Stephan, for the jungle in our flat, for being such an efficient researcher, and for explaining the scientific circus to me;

Zsuzsa, for sharing my enthusiasm for open problem sessions, mass spectrometry, and extensive walks, and for showing me how to spat and reconcile as good friends; and

Zürich, for Limmat and Zürisee, Kletterzentrum, Niederdörfli, Kunsthaus and amazing mountains.

Contents

Chapter 1

Introduction

1.1 Biological Motivation

Molecular biology has made tremendous progress since Watson and Crick proposed their double helix model for the DNA molecule in 1953: Today, the central dogma of molecular biology is well–established. DNA cloning and sequencing have become standard techniques, and complete genomes of several organisms, including humans, are available. The genetic code has been used to predict proteins from DNA sequences, and huge databases exist that contain hundreds of thousands of proteins, both hypothetical and real ones, together with their amino acid sequences and, if available, their function in the cell.

The rapid progress of molecular biology, especially in the last two decades, is strongly connected to the development of automated techniques for efficient data analysis. In fact, computational molecular biology, also referred to as bioinformatics, has evolved to a discipline of its own, and covers topics as diverse as sequence alignment, efficient search techniques in large databases, or prediction of three–dimensional protein structures. A general introduction to computational molecular biology can be found for instance in the books by Gusfield [43], by Pevzner [71], by Setubal and Meidanis [79], or by Waterman [91].

In this thesis, we will focus on combinatorial problems that arise in the realm of *digestion experiments* using enzymes. Enzymes are catalysts that can speed up chemical reactions in a cell. We focus on nucleases, that can cut DNA molecules, and proteases, that cleave proteins. Each enzyme type cuts at specific sequence patterns, the *restriction sites*. In the remainder of this introduction, we describe how nucleases can be used to cleave DNA, in particular in double and partial digestion experiments, and how proteins can be digested using proteases. Furthermore, we give a survey of the results that we present in this thesis.

1.2 Digesting DNA

A DNA molecule is a large molecule that is composed of smaller molecules, the nucleotides. There are four nucleotides, namely adenine (`A`), cytosine (`C`), guanine (`G`), and thymine (`T`), which form – very roughly speaking – sequences to build up DNA molecules. For our purposes, a DNA molecule is a string over the alphabet {`A, C, G, T`}.

A nuclease is an enzyme that can cleave DNA molecules at specific restriction sites. This process is called *digestion.* For instance, the enzyme EcoRI cuts each occurrence of the recognition pattern `GAATTC` in a DNA molecule into `G` and `AATTC`.[1] More than 3000 different nucleases are known, and their recognition patterns are usually sequences of four to eight letters.

Digestion Experiments

A digestion experiment for a DNA molecule works as follows. First, clones of the molecule are generated by replicating it many times. Then these clones are digested using a restriction enzyme. If the enzyme is applied for long enough, then it cuts at *all* restriction sites in each clone, yielding fragments between any two adjacent restriction sites. This process is called *full* or *complete digestion*, in contrast to *partial digestion*, where only very small amounts of the enzyme are used, or the enzyme is exposed for different amounts of time, such that we obtain all fragments between any two restriction sites (that do not need to be adjacent). In both cases, the number of nucleotides of a fragment can be measured by using gel electrophoresis, a standard technique in molecular biology. Here, the DNA fragments are placed on one side of a gel block. Since DNA molecules are charged, they start moving through the gel if they are exposed to an electric field. The distance a fragment travels in the gel is inversely proportional to the mass of the molecule, which is itself proportional to the number of nucleotides in the molecule. Hence, we can interpolate the *fragment length*, i.e., the number of nucleotides in a fragment, from the position of the fragment in

[1] In fact, DNA molecules form a double stranded helix, where each adenine is paired with a thymine, and each cytosine is paired with a guanine (Watson–Crick pairs). For this reason, if EcoRI cuts at `GAATTC` in one strand, then it cuts at `CTTAAG` in the other strand at the same time, and the recognition pattern often forms a palindrome, respecting the Watson–Crick pairs.

the gel after a certain amount of time. This leaves us with a multiset of fragment lengths, one for each fragment, which can be used to explore the structure of the DNA molecule, e.g. by physical mapping (see below). An example for full and partial digestion using a fictional enzyme that cuts each occurrence of `CG` into `C` and `G` is shown in Figure 1.1.

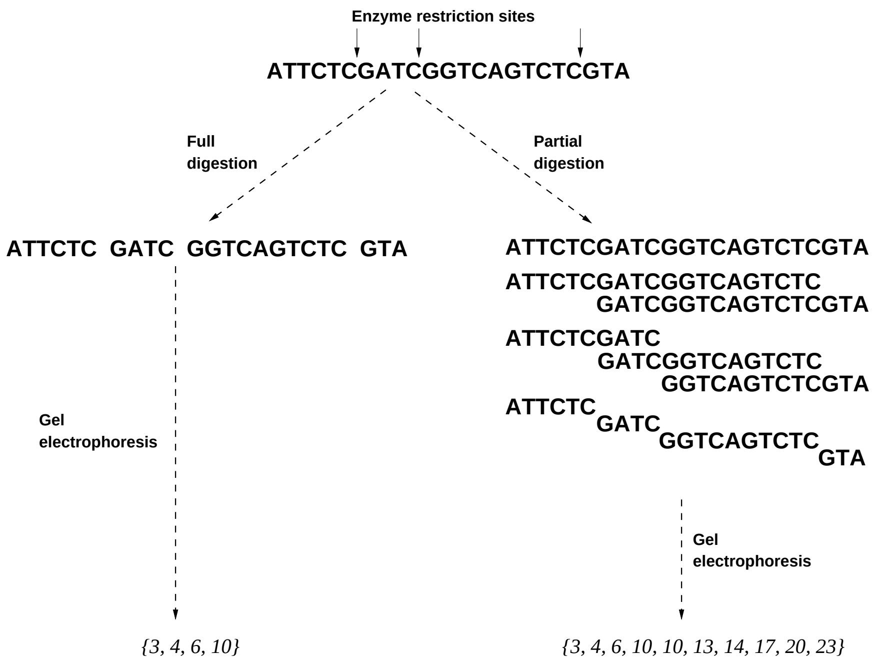


Figure 1.1: Full and partial digestion of DNA molecule `ATTCTCGATCGGTCA-GTCTCGTA` for an enzyme that cuts every pattern `CG` into `C` and `G`.

Digestion experiments can be used to construct *physical maps* of DNA molecules. A physical map describes the location of markers (in this case the restriction sites) along the DNA molecule. Physical maps are used for instance to find the appropriate positions of known fragments of a DNA molecule, even without sequencing the complete molecule. First successful restriction site mappings were performed in the 1970's [30, 83].

Double Digestion

The fragment lengths resulting from a single full digestion experiment cannot yield any information about the ordering of the fragments or the positions of the restriction sites, respectively, in an unknown DNA molecule. For this reason, *double digestion* experiments are performed, where *two* different enzymes are used as follows. First a set of clones of the DNA molecule is digested by an enzyme A; then a second set of clones is digested by another enzyme B; and finally, a third set of clones is digested by a mix of both enzymes A and B, which we will refer to as C. All digestions are full digestions. This results in three multisets of DNA fragments, and in three multisets of distances between *all adjacent restriction sites.* The objective is to reconstruct the original ordering of the fragments in the DNA molecule. This is referred to as the DOUBLE DIGEST problem. In the following definition of the DOUBLE DIGEST problem, $\text{sum}\,(S)$ denotes the sum of the elements in a multiset S, and $\text{dist}\,(P)$ is the multiset of all distances between two neighboring points in a set P of points on a line.

Definition 1.2.1 (DOUBLE DIGEST)**.** *Given three multisets A, B and C of positive integers with $\text{sum}\,(A) = \text{sum}\,(B) = \text{sum}\,(C)$, are there three sets P^A, P^B and P^C of points on a line, such that 0 is the minimal point in each set, $\text{dist}\,(P^A) = A$, $\text{dist}\,(P^B) = B$, $\text{dist}\,(P^C) = C$, and $P^A \cup P^B = P^C$?*

For example, given multisets $A = \{5, 15, 30\}$, $B = \{2, 12, 12, 24\}$ and $C = \{2, 5, 6, 6, 7, 24\}$ as an instance of DOUBLE DIGEST, then $P^A = \{0, 5, 20, 50\}$, $P^B = \{12, 14, 26, 50\}$ and $P^C = \{5, 12, 14, 20, 26, 50\}$ is a feasible solution, which is shown in Figure 1.2 (there may exist more solutions).

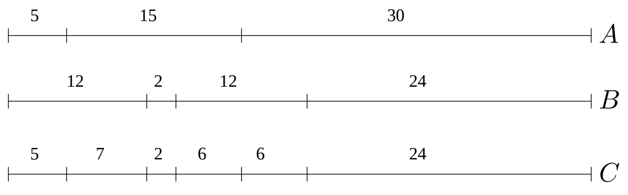


Figure 1.2: Example for the DOUBLE DIGEST problem.

The DOUBLE DIGEST problem is NP-complete, and several approaches, including exponential algorithms, heuristics, or computer–assisted interactive strategies, have been proposed (and implemented) in order to tackle the problem. We will study the DOUBLE DIGEST problem in Chapter 3. In particular, we will show that the problem becomes hard to approximate if the input data is prone to error.

Partial Digestion

A second approach to finding physical maps of DNA molecules is by *partial digestion* experiments. Here, we use only one enzyme to partially digest one set of clones, resulting in all fragments between *any two restriction sites*, respectively in the corresponding multiset of fragment lengths. Again, the objective is to reconstruct the original ordering of the fragments in the DNA molecule, which is referred to as PARTIAL DIGEST problem. A formal definition of this problem is as follows.

Definition 1.2.2 (PARTIAL DIGEST). *Given an integer m and a multiset D of $k = \binom{m}{2}$ positive integers, is there a set $P = \{p_1, \ldots, p_m\}$ of m points on a line such that $\{|p_i - p_j| \mid 1 \le i < j \le m\} = D$?*

For example, for the distance multiset $D = \{2, 5, 7, 7, 9, 9, 14, 14, 16, 23\}$, the point set $P = \{0, 7, 9, 14, 23\}$ is a feasible solution, which is shown in Figure 1.3 (there exist more solutions).

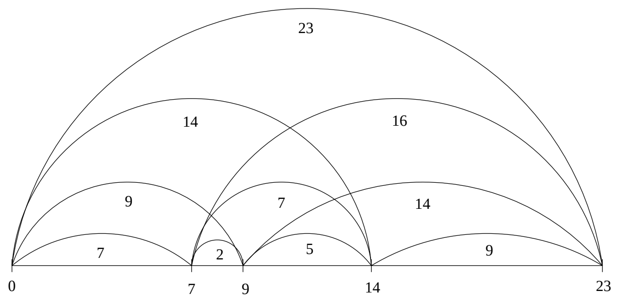


Figure 1.3: Example for the PARTIAL DIGEST problem.

The exact computational complexity of PARTIAL DIGEST is a long–standing open problem; in fact, in its pure combinatorial formulation it appears already in the 1930's in the area of X–ray crystallography. The problem can be solved in pseudo–polynomial time, and there exists a backtracking algorithm, for exact or erroneous data, which has expected running time polynomial in the number of distances, but exponential worst case running time. For the original PARTIAL DIGEST problem, neither a polynomial–time algorithm nor a proof of NP-completeness is known. In Chapter 4, we will show that PARTIAL DIGEST becomes hard to solve if the input data is prone to error, namely if there are missing or additional distances, or if length measurements are erroneous.

1.3 Digesting Proteins

Proteins are large molecules that are made up of smaller molecules, the amino acids, which are linked together by peptide bonds. There are 20

Amino Acid	**3–Letter Code**	**1–Letter Code**	**Monoisotopic Mass**	**Average Mass**
Alanine	Ala	A	71.03711	71.0788
Arginine	Arg	R	156.10111	156.1876
Asparagine	Asn	N	114.04293	114.1039
Aspartic Acid	Asp	D	115.02694	115.0886
Cysteine	Cys	C	103.00919	103.1448
Glutamic Acid	Glu	E	129.04259	129.1155
Glutamine	Gln	Q	128.05858	128.1308
Glycine	Gly	G	57.02146	57.0520
Histidine	His	H	137.05891	137.1412
Isoleucine	Ile	I	113.08406	113.1595
Leucine	Leu	L	113.08406	113.1595
Lysine	Lys	K	128.09496	128.1742
Methionine	Met	M	131.04049	131.1986
Phenylalanine	Phe	F	147.06841	147.1766
Proline	Pro	P	97.05276	97.1167
Serine	Ser	S	87.03203	87.0782
Threonine	Thr	T	101.04768	101.1051
Tryptophan	Trp	W	186.07931	186.2133
Tyrosine	Tyr	Y	163.06333	163.1760
Valine	Val	V	99.06841	99.1326

Figure 1.4: Amino acid codes and masses (acc. to [84, 101, 102]).

amino acids[2], and their sequence constitutes the primary structure of a protein.[3] For the purposes of this thesis, we will view a protein as a string over an alphabet of size 20, where each amino acid is represented by its single letter code (see Table 1.4). Protein sizes range from below 100 to several thousand amino acids, where a typical protein has length 300 to 600.

Proteomics is the field that investigates the proteins which are expressed (i.e., produced) at a certain time in a certain cell type. Due to the development of novel techniques in proteomics that allow for high–throughput experiments, large amounts of data are being accumulated in databases. For instance, SWISS–PROT contains information on approximately 100,000 proteins [11, 113], and PIR has even more than 200,000 entries [96, 110].

[2]Recently, two new amino acids have been discovered [85]; however, in this thesis we will refer only to the 20 amino acids that are most common.

[3]Higher order structures of proteins determine the positions of the amino acids in the three–dimensional space.

When a protein is isolated in an experiment, one would like to know whether it is already known, and if so, one would like to find the corresponding information in the databases. Otherwise, if the protein is new, then we will start to investigate it from scratch. Hence, fast algorithms for protein identification are required.

An obvious way to identify a protein is to establish its amino acid sequence. This is called *de novo protein sequencing*. However, standard protein sequencing, unlike DNA sequencing, is very expensive both in time and money. For instance, identifying *one* amino acid by Edman degradation, a standard method for protein sequencing, takes about 45 minutes, which makes this approach infeasible in a high–throughput context. Therefore, other methods are required for fast and reliable identification of proteins. In the second part of this thesis, we will study two different methods for protein identification that are both based on data obtained from enzyme digestion experiments, namely mass fingerprints and tandem mass spectra.

Mass Fingerprints

Similar to nucleases (which cut DNA molecules), proteases are enzymes that can cleave proteins, where one specific protease always cuts proteins at the same patterns of amino acids. For instance trypsin, one of the most frequently used proteases, cuts after each occurrence of arginine (`R`) or lysine (`K`), unless the next amino acid is proline (`P`). Digesting a protein results in a set of protein fragments, referred to as *peptides*. The masses of these fragments can be determined by *mass spectrometry*, which yields a multiset of peptide masses referred to as *mass fingerprint* of the protein.

For example, if a protein with sequence `VNGYSEIERFGMLGAARPAKEF` is digested by trypsin, then this results in peptides `VNGYSEIER`, `FGMLGAARPAK` and `EF`, and mass fingerprint $\{1065.49851, 1117.59604, 294.111\}$. Observe that the values in a mass fingerprint differ from the sums of the amino acid masses in the peptide sequence by $+18$, since each peptide has an additional hydrogen atom ($+1$) and an additional OH–group ($+17$) at its terminals.

For a short introduction to the main techniques in mass spectrometry, we refer the reader to the survey by Mann et al. [58]. More detailed introductions to mass spectrometry can be found for instance in the books by James [49] or by Snyder [84].

A commonly employed approach for protein identification without sequencing uses the mass fingerprint of a protein to look up databases of known proteins: If the breakup pattern of the enzyme is known, which is the case for most enzymes, then we can compute the theoretical mass fingerprint for each protein in a database, and compare it to the experimental fingerprint.[4] For each protein in the database this takes time linear in the length of the protein. If the breakup points are known *in advance*, e.g. if we know that we will always use trypsin to cleave the protein, then we can preprocess the database in an appropriate way to allow even for sublinear search time.

The problem to look up a database becomes more challenging if the breakup points are not known. This can be the case, for instance, when different proteases are used in the digestion step, or when the protein was subject to post–translational modifications such as phosphorylation or glycosylation, where an additional phosphorus or sugar molecule, respectively, attaches to some amino acids in the protein. If such modifications occur at restrictions sites where the enzyme would be supposed to cut, then cleavages at these sites are diminished (the sites are "blocked"), thus they may yield an experimental fingerprint that does not match the theoretical fingerprint of the protein. In the extreme, we can assume that nothing is known about the breaking points in the digestion process, i.e., we assume that the protein breaks at arbitrary positions. In this case, we ask for a protein in the database that matches the experimental fingerprint best, i.e., that has a maximum number of disjoint substrings whose sum of amino acid masses occur in the fingerprint (*submasses*). For our purposes, substrings are always contiguous. One obvious way to find such best matches in a database is to check for each single mass in the fingerprint whether it is a submass of a protein from the database, individually for each protein in the database. This yields the Mass Finding problem, which is formally defined as follows, where $\mathbb{N}$ denotes the set of positive integers.

Definition 1.3.1 (Mass Finding)**.** *Given an alphabet $\mathcal{A}$, a mass function $\mu : \mathcal{A} \to \mathbb{N}$, and a string σ over $\mathcal{A}$, find a data structure and a query algorithm which, for a given positive integer M, decides whether σ has a substring of mass M, where the mass of a string is the sum of the masses of its letters.*

String σ is referred to as *weighted string*, and mass M is also referred to as *weight.* We use integer masses in the definition of the Mass Finding problem, since the accuracy of mass measurements is restricted, and we can multiply the masses in a mass fingerprint by an appropriate factor, e.g. a power of 10, to obtain integers.

[4]This is an idealistic point of view: In fact, real–life data is always prone to error, and for biological applications methods are needed that are not only efficient, but also fault tolerant: They need to be tolerant to measurement errors, missing or additional masses in the fingerprint, and to sequencing errors of the database entries. However, for the purposes of this introduction we assume that we are given data without error.

In Chapter 6, we will study different algorithms for the MASS FINDING problem. In particular, we will present an algorithm that allows to answer queries in time sublinear in the length of σ, using a data structure that requires only linear space.

Tandem Mass Spectra

Identifying a protein using its mass fingerprint has become a powerful tool in protein analysis, and in fact it is today a standard technique in proteomics. However, this method depends upon the presence of the protein in question in the database, and if the protein cannot be found in the database, e.g. because it is an unknown protein, then this technique must fail. For this reason, other database independent techniques are required that allow to identify proteins. One such technique – that we study here – is *de novo peptide sequencing using tandem mass spectrometry.* This technique makes use of the differences in molecular weights of amino acids to determine the amino acid sequence of a peptide: First, the protein is digested using an enzyme such as trypsin, breaking it up into peptides (shorter amino acid sequences). Then these peptides are ionized, separated according to their mass using a mass spectrometer, and single peptides are further fragmented using collision induced dissociation (CID). In this dissociation step, ideally[5] each single peptide molecule breaks at one random position between two amino acids, resulting in two complementary ion types: b–ions, that correspond to prefixes of the amino acid sequence of the peptide, and y–ions, that correspond to suffixes. In fact, fragment ions occur that correspond to all prefixes and all suffixes of the amino acid sequence of the peptide. E.g. peptide `VNGYSEIER` can break up into fragments `V` and `NGYSEIER`, into fragments `VN` and `GYSEIER`, into fragments `VNG` and `YSEIER`, and so on.

The abundance of fragments of different masses are measured, again using mass spectrometry, which results in the *tandem mass spectrum* (or *MS/MS spectrum*) of the peptide.[6] E.g. peptide `VNGYSEIER` yields b-ions with masses 100.06841, 214.11134, 271.1328, 434.19613, 521.22816, 650.27075, 763.35481, 892.3974, 1048.4985, and y-ions with masses 175.10111, 304.1437, 417.22776, 546.27035, 633.30238, 796.36571, 853.38717, 967.4301, 1066.4985. Observe that these masses differ from the sum of amino acid masses of the corresponding prefixes respectively suffixes, since every b–ion has an additional terminal hydrogen atom (+1), while y–ions have an additional termi-

[5] We say "ideally" because a single molecule can – rather infrequently – break up into more than one fragment.

[6] To be exact, the *mass/charge ratios* of the fragments are measured in tandem mass spectrometry; however, we will speak of masses here, since in our setting, the charge state will be known, hence we can easily determine the mass from the ratio, and vice versa.

nal OH–group (+17), an additional hydrogen atom (+1), and an additional proton (+1).

MS/MS spectra can be visualized by a graph, where the x–axis shows the masses, and the y–axis their abundances. A tuple (mass, abundance) is referred to as *peak*. Figure 1.5 shows a spectrum for peptide `VNGYSEIER` in ASCII–format representation; the corresponding graphical representation is shown in Figure 1.6, and Figure 1.7 shows an annotated variant, where those peaks that correspond to b–ions and y–ions are marked. Observe that only few peaks in the spectrum correspond to peptide ions. We will refer to these peaks as *true* peaks, in contrast to *noise peaks*, that do not directly correspond to ions.

MS/MS spectra can be used for de novo peptide sequencing as follows: Two adjacent prefixes of a protein sequence differ by exactly one amino acid, and the corresponding masses in the MS/MS spectrum differ by the corresponding amino acid mass. The same holds for adjacent suffixes. Moreover, if we assume a noise–free spectrum, where only peaks occur that correspond to prefixes or suffixes, then for each prefix peak there is a corresponding suffix peak such that their masses sum up to the total peptide mass (up to a constant offset). Although we cannot distinguish between prefix peaks and suffix peaks directly from the data, there are algorithms available that can determine the correct peptide sequence efficiently, given a noise–free MS/MS spectrum. However, real–life MS/MS spectra are always prone to error; in fact, the number of noise peaks is often much larger than the number of true peaks in a spectrum, while on the other hand some true peaks can be missing. Hence, for real–life data the problem becomes challenging, since we have to identify the true peaks in a spectrum, and we have to "guess" which true peaks are missing. We have implemented a tool for de novo sequencing that uses diverse heuristics to identify true peaks. In Chapter 7, we present this tool, and discuss its performance on real–life data.

1.4 Overview and Summary of Results

Overview

In the following, we summarize the main results that we present in this thesis. For formal definitions, previous work, and more biological background, we refer the reader to the introductory sections in the corresponding chapters.

We first introduce some notation and basic definitions in Chapter 2. In particular, we recapitulate definitions of several combinatorial problems that we will use later on to prove NP-hardness respectively inapproximability

```
1066.91 2      405.6 3223.0   491.3 3661.0     605.4 1974.0     798.4 2931.0
155.4 1853.0   406.6 18276.0  493.6 1907.0     614.2 1638.0     830.5 2074.0
169.6 664.0    407.4 2527.0   494.5 4247.0     615.4 22736.0    835.0 920.0
175.2 8867.0   409.1 1526.0   501.4 915.0      616.3 5114.0     836.8 2661.0
176.1 1917.0   415.8 4421.0   502.8 2291.0     617.4 1172.0     839.5 691.0
193.4 1127.0   416.4 13908.0  504.1 6375.0     629.5 2225.0     850.0 1379.0
197.2 4675.0   417.3 84612.0  504.8 8175.0     632.4 10133.0    853.5 163042.0
198.3 1494.0   418.3 36599.0  505.6 5511.0     633.5 279256.0   854.5 94640.0
214.2 20325.0  419.6 13245.0  507.2 3384.0     634.5 76684.0    855.5 13190.0
215.4 7726.0   420.6 340.0    508.0 1528.0     635.5 10848.0    857.2 8845.0
221.5 2407.0   422.2 735.0    510.8 1071.0     647.3 5196.0     873.6 894.0
253.4 14425.0  427.4 1633.0   511.6 1490.0     647.9 1.0        875.0 16599.0
254.2 2864.0   430.8 1238.0   516.4 25188.0    650.3 12329.0    877.2 9267.0
271.3 10370.0  434.3 46597.0  517.3 90867.0    651.4 5034.0     878.3 549.0
287.5 2175.0   435.4 12917.0  518.4 32137.0    664.0 2640.0     892.3 17455.0
291.2 2482.0   436.0 637.0    519.5 15594.0    665.6 1214.0     893.4 12507.0
304.6 21919.0  437.2 758.0    520.3 3464.0     687.6 847.0      894.6 1001.0
305.4 2023.0   452.2 1216.0   521.4 3477.0     700.3 978.0      903.7 1234.0
306.5 983.0    457.3 984.0    522.4 5074.0     710.3 1203.0     914.3 2168.0
318.2 2034.0   458.4 11016.0  525.3 1221835.0  717.0 1168.0     915.4 1528.0
336.5 2423.0   459.5 4629.0   526.2 123797.0   718.3 1517.0     932.4 87420.0
340.2 2696.0   461.0 1639.0   527.5 7562.0     719.4 1326.0     933.4 42752.0
341.0 1309.0   465.2 1613.0   528.3 4651.0     720.4 1393.0     934.4 5974.0
354.8 998.0    466.4 4692.0   534.4 1627.0     721.4 3350.0     949.2 992.0
360.7 1154.0   467.4 16976.0  535.0 107375.0   728.3 4230.0     950.4 7076.0
363.6 2120.0   468.2 1646.0   535.8 11686.0    740.8 2375.0     951.3 4848.0
376.7 1030.0   469.3 3729.0   537.3 2356.0     745.3 5871.0     952.7 1429.0
381.9 1576.0   470.7 2645.0   546.5 48468.0    746.3 15350.0    960.6 564.0
387.5 6471.0   471.5 1927.0   547.5 13341.0    747.3 3212.0     967.5 14881.0
388.3 21433.0  475.4 2534.0   551.1 803.0      754.1 3677.0     968.3 11520.0
389.4 95073.0  477.4 813.0    558.3 1082.0     757.0 1601.0     969.6 2683.0
390.4 23115.0  485.0 36972.0  569.4 1859.0     763.3 9815.0     977.6 1276.0
391.2 5348.0   486.1 7971.0   587.4 4226.0     764.4 5140.0     979.7 547.0
398.9 2205.0   486.8 8986.0   588.6 2371.0     778.5 765.0      1021.7 899.0
399.8 977.0    487.5 2660.0   597.4 6254.0     789.3 1383.0
402.8 1741.0   488.3 3445.0   598.4 1427.0     795.6 718.0
403.8 1508.0   489.1 718.0    599.5 5644.0     796.5 69341.0
404.6 910.0    490.5 1708.0   604.2 526.0      797.5 39265.0
```

Figure 1.5: Example of a spectrum for peptide `VNGYSEIER`, represented as `.dta`–file (printed in 5 columns): The first line specifies the mass of the peptide, here 1066.91, and its charge state, here 2; each of the following lines denote a mass and its abundance.

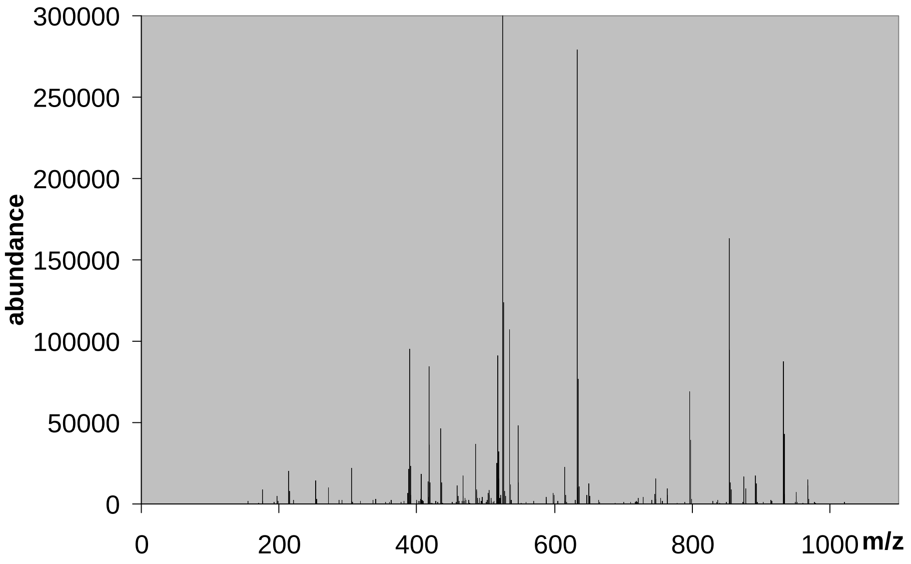


Figure 1.6: Graphical representation of the spectrum from Figure 1.5 for peptide `VNGYSEIER` (peak 525.3 is truncated).

results. In Chapters 3–5, we study the complexity of problems that arise in the realm of digestion experiments for DNA molecules, namely the problems Double Digest, Partial Digest and Equal Sum Subsets. In the second part of this thesis we address proteins instead of DNA, and study in Chapters 6–8 the problems Mass Finding and Decomposition, and present our de novo peptide sequencing tool Audens.

A summary of the results obtained in this thesis and final conclusions can be found in Chapter 9. In the Appendix, we present the experimental results for Audens. Moreover, we give a list of all combinatorial problems that are used throughout this thesis. The accompanying CD includes the Audens program, the test data used to evaluate Audens, and the thesis itself. The contents of the CD is shown in Appendix B.

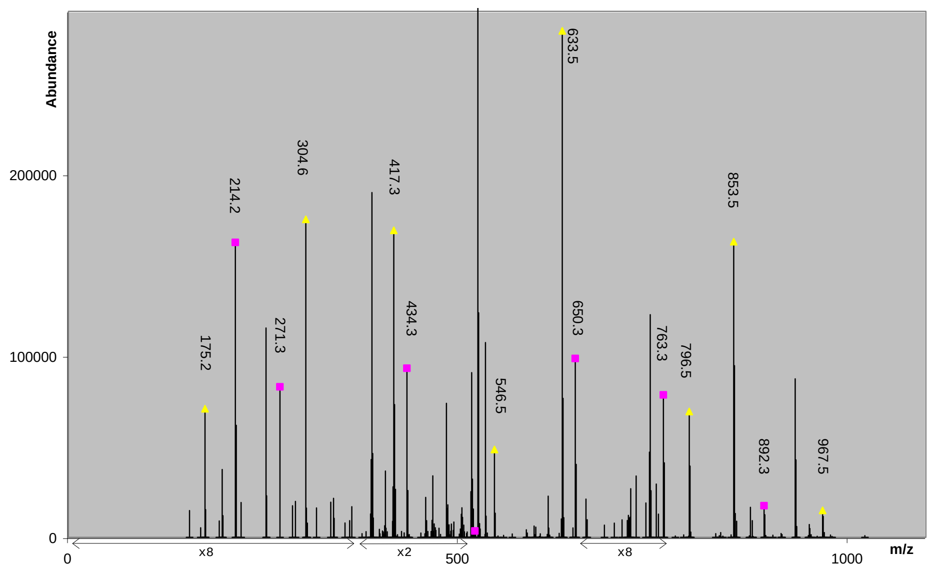


Figure 1.7: Annotated graphical representation for the spectrum from Figure 1.5. Part of the abundances are scaled by factor 2 respectively 8 for sake of presentation, denoted by "×2" respectively "×8". Peaks with squares correspond to b–ions, those with triangles to y–ions (peak 525.3 is truncated).

Double Digest

In Chapter 3, we study the complexity of combinatorial problems in the realm of Double Digest. The Double Digest problem is known to be NP-complete, and several approaches including exponential algorithms, heuristics, and computer–assisted interactive strategies have been proposed in the literature.

In real–life, double digest experiments are usually carried out with two enzymes that cut at different restriction sites. For this reason, we introduce the Disjoint Double Digest problem, which is equivalent to the Double Digest problem with the additional requirement that the two enzymes may never cut at the same site. As a first set of results we prove in Section 3.2 that both Double Digest and Disjoint Double Digest are NP-complete in the strong sense.

A *partial cleavage* error occurs in a double digestion experiment if an enzyme fails to cut at a restriction site where it would be supposed to cut. In Section 3.3, we discuss several models to measure partial cleavage errors, and study the corresponding optimization variations of Double Digest. We first show that the problem Min Absolute Error Double Digest, where we want to minimize the *absolute* number of partial cleavage errors $|(P^A \cup P^B) - P^C| + |P^C - (P^A \cup P^B)|$ in a solution, cannot be approximated to within any finite approximation ratio, unless $\mathsf{P} = \mathsf{NP}$. The situation becomes slightly better if we measure the amount of errors *relative* to the input size by adding offset $|A| + |B| + |C|$. We show that the corresponding minimization problem Min Relative Error Double Digest cannot be approximated to within factor $\frac{877}{876}$, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, we show that any arbitrary arrangement of the distances achieves an approximation ratio of 2.

We then study the Min Point Number Double Digest problem, where we measure the *total size* of a solution, i.e., where we want to minimize the total number of points $|P^A \cup P^B \cup P^C|$ in a solution. We show that this problem cannot be approximated to within factor $\frac{392}{391}$, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, we again show that any arbitrary arrangement of the distances achieves an approximation ratio of 3.

For each of our Double Digest optimization problems, a variation can be defined where the enzymes may only cut at disjoint restriction sites, analogous to Disjoint Double Digest. The corresponding optimization problems are Min Relative Error Disjoint Double Digest and Min Point Number Disjoint Double Digest. In Section 3.4, we show that these problems are even harder to solve than the unrestricted variants. In fact, we show that it is NP-hard to even find *feasible* solutions for instances of any (reasonable) optimization variation of Double Digest with disjoint restriction sites. To establish this result, we introduce the problem Disjoint Ordering, where we have to arrange two given sets of numbers in a disjoint fashion:

Definition 1.4.1 (Disjoint Ordering)**.** *Given two multisets A and B of positive integers with* $\operatorname{sum}(A) = \operatorname{sum}(B)$*, are there two sets P^A and P^B of points on a line, such that* 0 *is the minimal point in each set,* $\operatorname{dist}(P^A) = A, \operatorname{dist}(P^B) = B$ *and* $P^A \cap P^B = \{0, \operatorname{sum}(A)\}$*?*

We show that Disjoint Ordering is NP-hard. Any polynomial–time algorithm that claims to achieve a finite approximation ratio for any Dou-

BLE DIGEST optimization variation with disjoint restriction sites has to be able to find feasible solutions for all instances, which is just the DISJOINT ORDERING problem. Thus, no such algorithms can exist, unless $\mathsf{P} = \mathsf{NP}$.

Partial Digest

The PARTIAL DIGEST was – as a combinatorial problem – already proposed in the 1930's and has since then been subject of intensive research: There exist backtracking and pseudo–polynomial algorithms, polynomial bounds on the number of feasible solutions, and numerous other insights; however, the exact computational complexity of PARTIAL DIGEST is still open.

In partial digest experiments, typically four types of errors occur: *additional fragments*, for instance through contamination of the probe with unrelated biological material; *missing fragments* due to partial cleavage errors, or because of small fragments that remain undetected by gel electrophoresis; *incorrect fragment lengths*, due to the fact that fragment lengths cannot be determined exactly using gel electrophoresis; and, finally, *wrong multiplicities* due to the intrinsic difficulty to determine the proper multiplicity of a distance from the intensity of its band in a gel. In an effort to model real–life data, we introduce variations of PARTIAL DIGEST that model the first three error types.

We first study the computational complexity of the minimization problem MIN PARTIAL DIGEST SUPERSET, in which we model omissions: Only a subset of all pairwise distances is given – the rest are lacking due to experimental errors – and we ask for a minimum number of points on a line which cover all these distances. We show in Section 4.2 that this variation is NP-hard to solve exactly. This result answers an open problem proposed in the book by Pevzner [71].

The MAX PARTIAL DIGEST SUBSET problem models the situation of additions, where we are given data in which some wrong distances were added, and we search for a point set P of maximum cardinality such that all pairwise distances from P are covered by the input distances (or, expressed differently, such that the number of distances in the input that do not occur in the solution P is minimum). In Section 4.3, we show that this maximization problem is hard to approximate to within factor $|D|^{\frac{1}{2}-\varepsilon}$, for any $\varepsilon > 0$, unless $\mathsf{NP} = \mathsf{ZPP}$. Here, $|D|$ is the number of input distances. This inapproximability result is tight up to low–order terms, as we give a trivial approximation algorithm that achieves a matching approximation ratio.

Finally, in Section 4.4, we study the computational complexity of the variation of PARTIAL DIGEST where all distances are present, but each is prone to some additive error. If individual additive error bounds can be assigned to each distance, then it is known that the Partial Digest problem becomes strongly NP-hard. In the corresponding proof, the fact that even error bound zero can be assigned to some distances (and a non–zero error bound can be assigned to other distances) is heavily exploited. We study the more general problem variation where all distances are prone to the *same non–zero error*. More precisely, we introduce the problem Partial Digest With Errors, where we are given a multiset of distances and an error $\varepsilon > 0$, and we ask for a set of points on a line such that their pairwise distances match D, up to error at most $\pm\varepsilon$ for each individual distance. We show that this problem is strongly NP-complete.

Equal Sum Subsets

As mentioned above, we prove in Section 4.2 that Partial Digest becomes NP-hard if we are given only a subset of all pairwise distances (Min Partial Digest Superset). To establish this result we give a reduction from Equal Sum Subsets, which is the problem where we are given a set of positive integers, and we ask for two disjoint subsets of the numbers that add up to the same sum. More formally, this problem is defined as follows.

Definition 1.4.2 (Equal Sum Subsets)**.** *Given a set A of n positive integers, are there two disjoint non–empty subsets $X, Y \subseteq A$ such that* $\operatorname{sum}(X) = \operatorname{sum}(Y)$*?*

We do not allow multisets here, as the problem is trivially solvable if the same number exists more than once in the input. While Equal Sum Subsets is known to be NP-complete, only very few studies have investigated the complexity of its variations. Motivated by the connection between Equal Sum Subsets and Partial Digest, we study several natural variations of the problem in Chapter 5.

We first study the problem Factor–r Sum Subsets, where we need to find two subsets such that the ratio of their sums meets some given ratio r. For $r = 1$, this is Equal Sum Subsets. We prove in Section 5.2 that Factor–r Sum Subsets is NP-complete for any rational $r > 0$.

In Section 5.3, we study the problem k Equal Sum Subsets, in which we need to find k disjoint subsets of equal sum. For $k = 2$, this is again the Equal Sum Subsets problem. We first show that k Equal Sum Subsets is NP-complete. Then we consider the case that k is not part of the problem definition, but a fixed function in n. We give a pseudo–polynomial time algorithm for k Equal Sum Subsets for the case that k is a constant, denoted by $k = O(1)$. On the other side, we prove that the problem is strongly NP-complete for k linear in n, denoted by $k = \Omega(n)$. We then study the problem under the additional requirement that the subsets should be of equal cardinality. We present a polynomial–time algorithm for the problem kESS Of Cardinality c, where the cardinality is part of the problem definition (i.e., a constant). On the other hand, if the cardinality is part of the input or not specified at all, then the corresponding problems – called kESS Of Specified Cardinality and kESS Of Equal Cardinality – are NP-complete. However, we can modify the algorithm for k Equal Sum SubsetsMentioned above in a way such that it runs in pseudo–polynomial time for these two problems.

After that we come back to the case of *two* equal sum subsets (instead of k). In many situations, a solution for an Equal Sum Subsets instance has to fulfill additional requirements. In ESS With Exclusions, we ask for two equal sum subsets such that certain pairs of numbers are not allowed to appear in the same subset. This problem is NP-complete, since it is equivalent to Equal Sum Subsets if there are no excluded pairs. We give a pseudo–polynomial time algorithm for this variation. On the other hand, we show NP-completeness for the problem ESS With Enforced Element, where we enforce one of the input numbers to be used in a solution, and for the problem Alternating Equal Sum Subsets, where we are given pairs of numbers and have to use either none of the two numbers of one pair, or both, and then in different sets.

We then study again variations of Equal Sum Subsets where we restrict the cardinality of a solution. The case that the two subsets should have equal cardinality is a special case of kESS Of Equal Cardinality, hence, the corresponding results can be transferred. We show in addition that the problem ESS Of Different Cardinality, where we ask for two subsets of *different* cardinality, is NP-complete.

Finally, we consider the situation where we are given *two* input sets and ask for two subsets of these sets that have equal sum. This is the ESS From Two Sets problem. We show that this problem is again NP-complete, and that it remains NP-complete even if we restrict the choice of elements from the two sets to have identical indices, or disjoint indices, or disjoint indices that cover all possible indices, or the same number of indices.

Mass Finding

In the second part of this thesis, we address problems that arise in the realm of protein identification. We start in Chapter 6 with the Mass Finding problem, where we ask whether a mass M occurs as submass in a given protein. Among others, we present two simple algorithms for this problem:

The first one answers a query for mass M in time linear in n, the number of amino acids of the given protein, and needs no additional data structure; the second algorithm stores in a preprocessing step *all* submasses of the protein in a hash table (or a sorted array) of size at most quadratic in n, and runs queries in constant (logarithmic) time. Since both linear time and quadratic space are inefficient in the setting of large databases, we ask for algorithms for the Mass Finding problem that beat both bounds at the same time. In fact, we present an algorithm called Lookup that solves the Mass Finding problem with linear storage space and sublinear query time $O(\frac{n}{\log n})$. However, this algorithms only serves as a proof that the two efficiency requirements can be met at the same time, since it requires unreasonably large inputs to become efficient.

De Novo Sequencing

De novo peptide sequencing on the basis of tandem mass spectrometry data is one of the most challenging problems in proteomics. As a first step on the way towards a new automated de novo sequencing tool, we have developed and implemented a prototype called Audens that works as follows. In a preprocessing step, we assign a *relevance value* to each peak in the input spectrum, using a number of heuristics ("grass mowers"). For instance, the relevance of peak p is increased if isotope peaks $p+1$ and $p+2$ are present in the spectrum as well, since this indicates that the peak belongs to a peptide ion, hence is not a noise peak. We then use a sequencing algorithm that is a modification of a dynamic programming algorithm introduced by Chen et al. [14] to find peptide sequences that maximize the sum of relevances of the corresponding peaks in the spectrum. Finally, the best matching sequences are output in a ranked list. In Chapter 7, we describe the basic concepts of Audens, in particular, the grass mowers and the sequencing algorithm that we apply. To determine the performance of Audens on real–life data, we ran it for a test set of 266 spectra for which the correct peptide sequence is known. Audens lists the correct sequence for 79 of these spectra among its first three candidates. For comparison, Lutefisk, another sequencing tool, found the correct sequence for only 68 of these spectra.

Decomposition

In the sequencing algorithm that we apply in Audens (see above), we use a subroutine that decides for a given mass M whether there exists a sequence of amino acids whose masses add up to M. We solve this problem by precomputing all masses M, up to a certain upper bound, that can be decomposed into amino acids. While this solves the problem for the purposes of our de novo sequencing tool, it poses at the same time the question how this problem can be solved in general, and gives rise to the following definition.

Definition 1.4.3 (Decomposition). *Given n positive integers $c_1, \ldots, c_n$ and a positive integer M, are there non–negative integers $\lambda_1, \ldots, \lambda_n$ such that $\sum_{i=1}^{n} \lambda_i \cdot c_i = M$?*

In the case of de novo sequencing, the c_i's correspond to the amino acid masses, and M to the mass we are looking for. This problem is also referred to as Coin Change problem or Integer Knapsack problem, and it is known to be NP-complete. In Chapter 8, we study the computational complexity of Decomposition under various restrictions, where we measure the time complexity in the input length, i.e., in the number n of amino acid masses and the logarithm of the total mass M.

We first observe that the Decomposition problem can be solved in polynomial time if the *number* of amino acids is constant, which is in principle the case for the 20 most common amino acids. However, due to the huge set of possible post–translational modifications that can virtually change the masses of amino acids, it is also reasonable to consider n to be non–constant.

We then study how the *size* of the amino acid masses in the input affects the complexity of Decomposition. If the total mass M itself is "small", i.e., bounded by a polynomial in n, then a standard dynamic programming algorithm solves the Decomposition problem in polynomial time. For the opposite case, where all amino acid masses are small and the total mass M is arbitrarily large, we give an algorithm that solves the problem in polynomial time as well. Then we extend this algorithm and show that the Decomposition problem can be solved in polynomial time even in the presence of few large amino acid masses , i.e., if there are few amino acids (e.g. a constant number) that have mass super–polynomial in n.

In the second part of Chapter 8, we study the complexity and approximability of the optimization variations Min Decomposition and Max Decomposition, where we ask for decompositions with a minimum respectively maximum number of amino acids (if a decomposition exists at all). We show for both problems that no polynomial time algorithm can exist that has constant approximation ratio, unless $\mathsf{P} = \mathsf{NP}$.

Chapter 2

Notation and Definitions

2.1 Introduction

In this chapter, we introduce the notation and basic concepts that we will use throughout this thesis. We first fix some notation and describe a vector representation for large numbers in Section 2.2. In Section 2.3, we sketch some concepts from complexity and approximability theory. Finally, in Section 2.4 we recapitulate the definitions of several combinatorial problems from the literature (like PARTITION), together with some known hardness and approximability results. We will use these results in our hardness and inapproximability proofs.

2.2 Notation

We do not distinguish between sets and multisets in our notation, and denote a multiset with elements $1, 1, 3, 5, 5$, and 8 by $\{1, 1, 3, 5, 5, 8\}$. Subtracting an element from a multiset will remove it only once (if it is there), thus $\{1, 1, 3, 5, 5, 8\} - \{1, 4, 5, 5\} = \{1, 3, 8\}$. Given a set or multiset S, then $|S|$ denotes the cardinality of S, e.g. $|\{1, 1, 3, 5, 5, 8\}| = 6$. By $\operatorname{sum}(S)$ we denote the sum of all elements in a set or multiset S, i.e., $\operatorname{sum}(S) = \sum_{x \in S} x$. E.g. $\operatorname{sum}(\{1, 1, 3, 5, 5, 8\}) = 23$.

By $\mathbb{Z}$ we denote the set of all integers, while $\mathbb{N}$ denotes the set of positive integers without 0.

For two points x and y in the plane we denote the Euclidean distance between x and y by $|x - y|$. Let $P = \{p_1, \ldots, p_n\}$ be a set of points on the real line, with $p_1 \leq \ldots \leq p_n$. We define the *distance multiset* of P by $\Delta(P) := \{|p_i - p_j| \mid 1 \leq i < j \leq n\}$, and say that P *covers* a multiset D

if $D \subseteq \Delta(P)$. By $\text{dist}(P)$ we denote the multiset of all distances between two *neighboring* points in P, i.e., $\text{dist}(P) = \{|p_{i+1} - p_i| \mid 1 \leq i \leq n-1\}$.

We introduce a vector representation for large numbers that will allow to add up the numbers digit by digit, like polyadic numbers. The numbers are expressed in the number system of some base Z. We denote by $\langle a_1, \ldots, a_n \rangle$ the number $\sum_{1 \leq i \leq n} a_i Z^{n-i}$; we say that a_i is the i-th digit of this number. In our proofs, we will choose base Z large enough such that the additions that we will perform do not lead to carry–overs from one digit to the next. Hence, we can add numbers digit by digit. The same holds for scalar multiplications. For example, having base $Z = 27$ and numbers $\alpha = \langle 3, 5, 1 \rangle, \beta = \langle 2, 1, 0 \rangle$, then $\alpha + \beta = \langle 5, 6, 1 \rangle$ and $3 \cdot \alpha = \langle 9, 15, 3 \rangle$. We define the concatenation of two numbers by $\langle a_1, \ldots, a_n \rangle \circ \langle b_1, \ldots, b_m \rangle := \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle$, i.e., $\alpha \circ \beta = \alpha Z^m + \beta$, where m is the number of digits in β. Let $\Delta_n(i) := \langle 0, \ldots, 0, 1, 0, \ldots, 0 \rangle$ be the number that has n digits, all 0's except for the i–th position, where the digit is 1. Moreover, $\mathbb{1}_n := \langle 1, \ldots, 1 \rangle$ has n digits, all 1's, and $\mathbb{0}_n := \langle 0, \ldots, 0 \rangle$ has n zeros. Notice that $\mathbb{1}_n = Z^n - 1$.

2.3 Concepts from Complexity and Approximability Theory

Complexity

We now recapitulate some concepts from complexity and approximability theory that we will use throughout this thesis. A more detailed discussion of computational complexity can be found for instance in the book by Garey and Johnson [40], while the books by Hochbaum, by Ausiello et al. and by Wegener give an introduction to (in–) approximability [3, 46, 92]. An online compendium of approximability results is maintained by Crescenzi and Kann and can be found in [106].

We say that an algorithm has *pseudo–polynomial running time* if the running time is polynomial in the input size for the case that all numbers in the input instance are coded unary. Expressed differently, the running time is bounded polynomially in the input size *and* in the largest number occurring in the input (in contrast to polynomial running times, which are bounded *only* in the input size). For instance, there exists an algorithm for Subset Sum that has pseudo–polynomial running time $O(n \cdot S)$, where n is the number of integers in the input, and S is the sum we are looking for [40]. Observe that this running time is not polynomial in n, unless we restrict S to be polynomial in n. Of course, if there are no numbers in the input of a problem, like in 3–Satisfiability, then pseudo–polynomial running times do not differ from polynomial running times; hence, we will speak of pseudo–polynomial running times only in the context of problems like Subset Sum, where numbers occur in the input.

A problem Π is NP-*hard in the strong sense*, or *strongly* NP-*hard*, if no algorithm can exist that solves the problem in pseudo–polynomial running time (unless $\mathsf{P} = \mathsf{NP}$). We can prove strong NP-hardness for a problem Π by giving a polynomial reduction to the problem Π from a problem Π' that is itself strongly NP-hard, e.g. 3–Partition or 3–Satisfiability, such that the reduction creates only instances in which all numbers are polynomially bounded in the size of the input instance of Π.

Approximability

Let Π be a maximization problem. The *approximation ratio* of an algorithm $\mathcal{A}$ *for instance* I is $\frac{OPT(I)}{\mathcal{A}(I)}$, where $\mathcal{A}(I)$ is the value of the objective function of the solution generated by algorithm $\mathcal{A}$ for instance I (the objective value), and $OPT(I)$ is the objective value of an optimum solution. The *approximation ratio of* $\mathcal{A}$ is the maximum approximation ratio for any instance I. The approximation ratio for minimization problems is defined by $\frac{\mathcal{A}(I)}{OPT(I)}$. In the following, we consider only maximization problems; the corresponding definitions for minimization problems are analogous.

In a *promise problem* variation of maximization problem Π we are promised that the objective value of an optimum solution for any instance I is either at least $U(I)$ or strictly less than $L(I)$, with $U(I) < L(I)$, and we have to decide which of the two cases is true. Such problems are also called *gap–problems*. For several NP-hard optimization problems it is known that the corresponding promise problems are still NP-hard for specific upper and lower bounds. For instance, for the problem Max Clique, where we are given a graph $G = (V, E)$ with n vertices and we ask for the maximum cardinality of a clique in G, the promise problem with upper bound $U(I) = k$ and lower bound bound $L(I) = \frac{k}{n^{\frac{1}{2}-\varepsilon}}$ is NP-hard to decide for any $0 < \varepsilon < \frac{1}{2}$ and suitable integer $k \leq n$ [44].

If a promise problem variation of a maximization problem Π is NP-hard for bounds $L(I)$ and $U(I)$, then this implies that no polynomial–time algorithm for Π can achieve an approximation ratio of $\frac{U(I)}{L(I)}$, unless $\mathsf{P} = \mathsf{NP}$. To see this, assume that there is a polynomial–time algorithm $\mathcal{A}$ with approximation ratio $R(I) \leq \frac{U(I)}{L(I)}$. We can use this algorithm to decide the promise problem of Π in polynomial time as follows. Given an instance I of the promise problem of Π, we interpret I as an instance of Π itself, and apply algorithm $\mathcal{A}$ to this instance. If the objective value of solution $A(I)$ is less than $L(I)$, then a maximum solution cannot be too large; in fact, we have $\frac{OPT(I)}{\mathcal{A}(I)} = R(I) \leq \frac{U(I)}{L(I)}$ from the definition of the approximation ratio of $\mathcal{A}$, hence $OPT(I) \leq \mathcal{A}(I)\frac{U(I)}{L(I)} < L(I)\frac{U(I)}{L(I)} = U(I)$. This yields the correct answer for the promise problem, since the objective value of a maximum solution by definition cannot be within the gap between $L(I)$ and $U(I)$. On the other hand, if $\mathcal{A}(I) \geq L(I)$, then this implies immediately that a maximum solution for I has at least objective value $U(I)$, since again the objective value cannot be between $L(I)$ and $U(I)$ by definition of the promise problem.

Let Π and Π' be two maximization problems. A *gap–preserving reduction* with parameters (c, ρ) and (c', ρ') transforms an instance I of Π into an instance I' of Π' in polynomial time such that the following two implications hold:

$$\begin{aligned} &\text{If } OPT(I) \geq c, \text{ then } OPT(I') \geq c'. \\ &\text{If } OPT(I) < \frac{c}{\rho}, \text{ then } OPT(I') < \frac{c'}{\rho'}. \end{aligned} \tag{2.1}$$

Here, c and ρ are functions in the size of I, and c' and ρ' are functions in the size of I', with $\rho(I), \rho'(I') \geq 1$. In the following we observe that gap–preserving reductions can be used to transfer NP-hardness from one promise problem to the other (for a proof, see e.g. [47]).

Fact 2.3.1. *Given two maximization problems Π and Π' and a gap–preserving reduction from Π to Π' with parameters (c, ρ) and (c', ρ'). If the promise problem of Π with upper bound $U(I) = c$ and lower bound $L(I) = \frac{c}{\rho}$ is NP-hard to decide, then the promise problem of Π' with upper bound $U(I') = c'$ and lower bound $L(I') = \frac{c'}{\rho'}$ is NP-hard.*

The existence of a gap–preserving reduction from Π to Π' implies immediately that if the promise problem of Π is NP-hard, then no approximation algorithm can exist for the optimization problem Π' that achieves an approximation ratio of ρ', unless $\mathsf{P} = \mathsf{NP}$. In this case, we say that Π is *hard to approximate to within factor ρ'*.

The class APX contains all optimization problems Π such that, for some $\rho > 1$, there is a polynomial–time algorithm for Π with approximation ratio ρ. Problem Π' is APX-*hard* if every problem Π from APX can be reduced to Π' by using an approximation preserving reduction (for a definition of approximation preserving reductions, see e.g. [3]).

2.4 Problems from the Literature

Throughout this thesis we will prove NP-hardness or inapproximability for several problems. In the proofs, we will give reductions from combinatorial problems from the literature that are known to be hard to solve or approximate. We recapitulate the definitions of these problems here, together with the corresponding hardness results.

Definition 2.4.1 (3–Partition)**.** *Given $3n$ positive integers $q_1, \dots, q_{3n}$ and an integer h such that $\sum_{i=1}^{3n} q_i = nh$ and $\frac{h}{4} < q_i < \frac{h}{2}$, for $i \in \{1, \dots, 3n\}$, are there n disjoint triples of q_i's such that each triple adds up to h?*

The 3–Partition problem is NP-complete in the strong sense [40]. Observe that $\frac{h}{4} < q_i < \frac{h}{2}$ already implies that each subset of the q_i's that adds up to h must have exactly three elements.

Definition 2.4.2 (Max 3–Dimensional Matching)**.** *Given three disjoint sets of positive integers W, X and Y of equal cardinality and a set $T \subseteq W \times X \times Y$, find a subset $M \subseteq T$ of maximum cardinality such that no two elements in M agree in any coordinate.*

The problem Max 3–Dimensional Matching is APX-hard [3] and hard to approximate to within factor $\frac{95}{94}$, unless $\mathsf{P} = \mathsf{NP}$ [16].

Definition 2.4.3 (Max Clique)**.** *Given a graph $G = (V, E)$ with vertices V and edges E, find a maximum clique in G, i.e., a maximum complete subgraph of G.*

The Max Clique problem is hard to approximate to within factor $n^{1-\varepsilon}$ for any $\varepsilon > 0$, unless $\mathsf{NP} = \mathsf{ZPP}$, where n is the number of vertices in G [44].

Definition 2.4.4 (Exact 3–Satisfiability)**.** *Given a set of m clauses $c_1, \dots, c_m$ over n Boolean variables $x_1, \dots, x_n$ such that each clause contains three positive literals, is there a (satisfying) assignment for the variables that satisfies exactly one literal per clause?*

The problem Exact 3–Satisfiability, which is also called One–in–Three 3–Satisfiability, is NP-complete [40].

Definition 2.4.5 (Partition)**.** *Given a set of n positive integers A, is there a subset $X \subseteq A$ such that $\operatorname{sum}(X) = \operatorname{sum}(A - X)$?*

Like Subset Sum, the Partition problem is NP-complete, but can be solved in pseudo–polynomial time $O(n \cdot \operatorname{sum}(A))$ [40].

Definition 2.4.6 (Subset Sum). *Given a set of n positive integers P and a number S, is there a subset $X \subseteq P$ such that* $\text{sum}(X) = S$*?*

The Subset Sum problem is NP-complete, but can be solved in pseudo–polynomial time $O(n \cdot S)$ [40].

Definition 2.4.7 (Alternating Partition). *Given n pairs of positive integers $(u_1, v_1), \dots, (u_n, v_n)$, are there two disjoint sets of indices I and J with $I \cup J = \{1, \dots, n\}$ such that $\sum_{i \in I} u_i + \sum_{j \in J} v_j = \sum_{i \in I} v_i + \sum_{j \in J} u_j$?*

The problem Alternating Partition, which is a variation of Partition, is NP-complete [40].

Chapter 3

Double Digestion

3.1 Introduction

In this chapter, we study the Double Digest problem, where we are given three multisets of distances and we ask for points on a line such that they cover all these distances. We recapitulate the definition from the introduction (cf. Definition 1.2.1):

Definition. *Given three multisets A, B and C of positive integers with* $\operatorname{sum}(A) = \operatorname{sum}(B) = \operatorname{sum}(C)$, *are there three sets P^A, P^B and P^C of points on a line, such that* 0 *is the minimal point in each set,* $\operatorname{dist}(P^A) = A, \operatorname{dist}(P^B) = B$, $\operatorname{dist}(P^C) = C$, *and $P^A \cup P^B = P^C$?*

Due to its importance in molecular biology, the Double Digest problem has been subject of intense research since the first successful restriction site mappings in the early 1970's [30, 83]: The Double Digest problem is NP-complete [41], and several approaches, including exponential algorithms, heuristics, additional experiments, and computer–assisted interactive strategies, have been proposed and implemented in order to tackle the problem [2, 8, 48, 52, 95]. The number of feasible maps for a Double Digest instance can be exponential in the number of fragments [41]. However, some maps can be transformed into each other using cassette transformations, and the set of different maps for an instance – modulo cassette transformations – can be characterized by using alternating Eulerian paths in appropriate graph classes [61, 70, 76]. For more information on the Double Digest problem, see for instance the books by Pevzner [71] and by Waterman [91].

The double digest experiment is usually carried out with two enzymes that cut DNA molecules at different restriction sites. For example, nuclease

BalI cleaves each occurrence of string `TGGCCA` into two substrings `TGG` and `CCA`, while nuclease SalI cuts each occurrence of string `GTCGAC` into two substrings `G` and `TCGAC`. In this case, the two enzymes never cut at the same site. A majority of all possible enzyme pairings of the more than 3000 known enzymes are pairs with such disjoint cutting behavior. On the other hand, some results in the literature rely on enzymes that cut at the same site in some cases (coincidences) [61]. In particular, NP-hardness of the Double Digest problem has so far only been shown using enzymes that allow for coincidences [41, 79, 91]. Indeed, such enzyme pairs exist. For example enzyme HaeIII cuts each `GGCC` string into `GG` and `CC`, and thus cleaves at a superset of the sites at which enzyme BalI cuts. However, having two enzymes that are guaranteed to always cut at disjoint sites seems more natural and might lead – at least intuitively – to easier reconstruction problems. For example, such instances always fulfill $|C| = |A| + |B| - 1$ (recall that $|S|$ denotes the cardinality of a multiset S). To reflect these different types of experiments, we define the Disjoint Double Digest problem, which is equivalent to the Double Digest problem with the additional requirement that the two enzymes may never cut at the same site, or, equivalently, that P^A and P^B are disjoint except for the first point (which is 0) and the last point (which is $\text{sum}(A) = \text{sum}(B)$).

Definition 3.1.1 (Disjoint Double Digest)**.** *Given three multisets A, B and C of positive integers with $\text{sum}(A) = \text{sum}(B) = \text{sum}(C)$, are there three sets P^A, P^B and P^C of points on a line such that 0 is the minimal point in each set, $\text{dist}(P^A) = A, \text{dist}(P^B) = B$, $\text{dist}(P^C) = C$, $P^A \cup P^B = P^C$, and $P^A \cap P^B = \{0, \text{sum}(A)\}$?*

The NP-hardness results for Double Digest in the literature [41, 79, 91] rely on reductions from weakly NP-complete problems (namely Partition). As a first set of results, we prove in Section 3.2 that both Double Digest and Disjoint Double Digest are actually NP-complete in the strong sense, by giving reductions from 3–Partition.

In Section 3.3, we try to model reality more closely by taking into account that double digestion data usually contains errors. As a matter of fact, all data in double digestion experiments is prone to error. Here, we consider *partial cleavage errors*, where an enzyme can fail to cut at some restriction site. Then one large fragment occurs in the data instead of two, or even more, smaller fragments. Such errors can occur for many reasons, e.g. improper reaction conditions or inaccurate DNA concentration (see for instance [111] for a list of possible causes). A partial cleavage error occurs for instance when an enzyme fails to cut at a site where it is supposed to cut in the first (second) stage of the double digest experiment, but then does cut at this site in the third phase, where it is mixed with the other enzyme. Such an error usually will make it impossible to find a solution for the corresponding DOUBLE DIGEST instance. In fact, only $P^A \cup P^B \subseteq P^C$ can be guaranteed for any solution. Vice–versa, if an enzyme cuts only in the first (second) phase, but fails to cut in the third phase, then we can only guarantee $P^C \subseteq P^A \cup P^B$.

In the presence of errors, usually the data is such that no exact solutions can be expected. Therefore, optimization criteria are necessary in order to compare and gauge solutions. We will define optimization variations of the DOUBLE DIGEST problem taking into account different optimization criteria; our objective will be to find good approximation algorithms. Obviously, an optimal solution for a problem instance with no errors will be a solution for the DOUBLE DIGEST problem itself.[1] Thus, the optimization problems cannot be computationally easier than the original DOUBLE DIGEST problem, and (strong) NP-hardness results for DOUBLE DIGEST carry over to the optimization variations.

In this chapter, we present several inapproximability results for optimization variations of DOUBLE DIGEST. These results only hold unless $\mathsf{P} = \mathsf{NP}$. For sake of readability, we refrain from mentioning this fact explicitely in the remainder of this chapter.

An obvious optimization criterion for DOUBLE DIGEST is to minimize the *absolute number* of partial cleavage errors in a solution, i.e., to minimize $|(P^A \cup P^B) - P^C| + |P^C - (P^A \cup P^B)|$. Here, points in $(P^A \cup P^B) - P^C$ correspond to errors where enzyme A or B failed to cut in the third phase of the experiment, and points in $P^C - (P^A \cup P^B)$ correspond to errors where enzyme A or B failed to cut in the first respectively second phase. The corresponding optimization problem MIN ABSOLUTE ERROR DOUBLE DIGEST, in which we try to find point sets P^A, P^B and P^C such that the absolute error is minimum, is formally defined as follows.

Definition 3.1.2 (MIN ABSOLUTE ERROR DOUBLE DIGEST). *Given three multisets A, B and C of positive integers such that $\text{sum}(A) = \text{sum}(B) = \text{sum}(C)$, find three sets P^A, P^B and P^C of points on a line such that 0 is the minimal point in each set, $\text{dist}(P^A) = A, \text{dist}(P^B) = B$, $\text{dist}(P^C) = C$, and $e(P^A, P^B, P^C) := |(P^A \cup P^B) - P^C| + |P^C - (P^A \cup P^B)|$ is minimum.*

We show in Section 3.3 that MIN ABSOLUTE ERROR DOUBLE DIGEST cannot be approximated to within any finite approximation ratio. This follows immediately from the fact that instances of DOUBLE DIGEST, if seen as an instance of our optimization problem, have optimum solutions with error 0.

[1]Of course, this only holds if the optimization criterion is well–designed.

We obtain a more sensible optimization criterion by measuring the amount of error *relative* to the input size, by adding $|A| + |B| + |C|$ as an offset to the number of errors. This yields the following problem definition.

Definition 3.1.3 (Min Relative Error Double Digest). *Given three multisets A, B and C of positive integers such that* $\operatorname{sum}(A) = \operatorname{sum}(B) = \operatorname{sum}(C)$*, find three sets P^A, P^B and P^C of points on a line such that* 0 *is the minimal point in each set,* $\operatorname{dist}(P^A) = A, \operatorname{dist}(P^B) = B,$ $\operatorname{dist}(P^C) = C$*, and $r(P^A, P^B, P^C) := |A| + |B| + |C| + e(P^A, P^B, P^C)$ is minimum.*

We show that Min Relative Error Double Digest cannot be approximated to within factor $\frac{877}{876}$. On the other hand, the problem can be approximated with factor 2, as we show that *any* arbitrary arrangement of the distances yields already a solution that is at most a factor 2 off the optimum. To show the non–approximability result, we give a gap–preserving reduction from Max 3–Dimensional Matching to Min Relative Error Double Digest.

As a third optimization criterion, instead of counting the number of errors, we measure the *total size* of a solution, which is a very natural optimization criterion, even if it does not model cleavage errors exactly. In this case, we want to minimize the total number of points in a solution, i.e., to minimize $|P^A \cup P^B \cup P^C|$. This yields the Min Point Number Double Digest problem, which is defined as follows.

Definition 3.1.4 (Min Point Number Double Digest). *Given three multisets A, B and C of positive integers such that* $\operatorname{sum}(A) = \operatorname{sum}(B) = \operatorname{sum}(C)$*, find three sets P^A, P^B and P^C of points on a line such that* 0 *is the minimal point in each set,* $\operatorname{dist}(P^A) = A, \operatorname{dist}(P^B) = B,$ $\operatorname{dist}(P^C) = C$*, and $|P^A \cup P^B \cup P^C|$ is minimum.*

We show that Min Point Number Double Digest cannot be approximated to within factor $\frac{392}{391}$. In the proof, we use basically the same techniques as for Min Relative Error Double Digest (in fact, we present the proof for Min Point Number Double Digest first). On the other hand, we show that *any* arbitrary arrangement of the distances yields a solution that is at most a factor of 3 off the optimum.

For each optimization problem of Double Digest, a variation can be defined where the enzymes may only cut at disjoint restriction sites, thus yielding equivalent variations of Disjoint Double Digest. In Section 3.4, we study these variations and show that – rather surprisingly – they are even harder than the unrestricted problems: It is NP-hard to even find a *feasible* solution for a given instance. To establish this result we show that the problem Disjoint Ordering, where we have to arrange two sets of numbers such that they do not intersect, is already NP-hard. Given an instance of any optimization variation of Disjoint Double Digest, it is obvious that the distances from A and B are arranged without intersections in any feasible solution, which is just the Disjoint Ordering problem. Hence, no polynomial–time algorithm can find feasible solutions, and thus, no finite approximation ratio can be achieved for any Disjoint Double Digest variation. This result does not only hold for the optimization criteria that we consider in this thesis, but it holds as well for any (reasonable) optimization variation of Disjoint Double Digest, since the proof does not depend on the optimization measure, but only on the requirement of disjointness.

Part of the results in this chapter have been published previously [23, 22].

3.2 Strong NP-Completeness of Double Digest and Disjoint Double Digest

In this section, we show strong NP-completeness for the decision problems Double Digest and Disjoint Double Digest. To this end, we present reductions from 3–Partition (see Definition 2.4.1).

We first extend the NP-completeness result from [41] for the Double Digest problem.

Theorem 3.2.1. Double Digest *is strongly* NP*-complete.*

Proof: The Double Digest problem is obviously in NP. To show strong NP-hardness we reduce 3–Partition, which is NP-complete in the strong sense [40], to Double Digest as follows: Given an instance $q_1, \ldots, q_{3n}$ and h of 3–Partition, let $a_i = c_i = q_i$, for $1 \leq i \leq 3n$, and let $b_j = h$ for $1 \leq j \leq n$. The three multisets A, B and C of a_i's, b_j's and c_i's, respectively, are an instance of Double Digest. If there is a solution for the 3–Partition instance, then there exist n disjoint triples of q_i's (or a_i's, respectively) such that each triple sums up to h. Starting from 0, we arrange the distances a_i on a line such that each three a_i's that correspond to the same triple are adjacent. The same ordering is used for the c_i's. This yields a solution for the Double Digest instance.

On the other hand, if there is a solution for the Double Digest instance, say P^A, P^B and P^C, then there exist n subsets of c_i's such that each subset sums up to h, since each point in P^B must occur in P^C as well, and all adjacent points in P^B have distance h. Then each of these subsets has exactly three elements, since $\frac{h}{4} < q_i < \frac{h}{2}$ by definition. Thus, these subsets yield a solution for the 3–Partition instance.

□

In the following, we show that Double Digest is strongly NP-complete even if we restrict it to enzymes that cut at disjoint restriction sites, which is the Disjoint Double Digest problem.

Theorem 3.2.2. Disjoint Double Digest *is strongly* NP*-complete.*

Proof: Disjoint Double Digest is obviously in NP. We show strong NP-hardness by reduction from 3–Partition. Given an instance $q_1, \ldots, q_{3n}$ and h of 3–Partition, let $s = \sum_{i=1}^{3n} q_i$ and $t = (n+1) \cdot s$. Recall that $s = nh$. We construct an instance of Disjoint Double Digest as follows. Let

$$\begin{aligned}
a_i &= q_i && \text{for } 1 \leq i \leq 3n, \\
\hat{a}_j &= 2t && \text{for } 1 \leq j \leq n-1, \\
b_j &= h + 2t && \text{for } 1 \leq j \leq n-2, \\
\hat{b}_k &= h + t && \text{for } 1 \leq k \leq 2, \\
c_i &= q_i && \text{for } 1 \leq i \leq 3n, \text{ and} \\
\hat{c}_j &= t && \text{for } 1 \leq j \leq 2n-2.
\end{aligned}$$

Let multiset A consist of the a_i's and $\hat{a}_j$'s, B consist of the b_j's and $\hat{b}_k$'s, and C consist of the c_i's and $\hat{c}_j$'s. Then $\text{sum}(A) = \text{sum}(B) = \text{sum}(C) = s + (2n-2) \cdot t$, and multisets A, B and C are an instance of Disjoint Double Digest.

If there is a solution for the 3–Partition instance, then there exist n disjoint triples of q_i's such that each triple sums up to h. Assume w.l.o.g. that the q_i's, and thus the a_i's and the c_i's, are ordered such that the three elements of each triple are adjacent. Starting in 0, we arrange the distances from A on a line such that each three a_i's that belong to the same triple are adjacent, and such that each three a_i's are separated by one $\hat{a}_j$ (see Figure 3.1). Let P^A be the corresponding point set. The distances from B are ordered $\hat{b}_1, b_1, \ldots, b_{n-2}, \hat{b}_2$, and P^B is the corresponding point set, starting in 0. For the distances c_i we use the same ordering as for the distances a_i, and each three c_i's are separated by two $\hat{c}_j$'s. Let P^C be the corresponding point set. Then P^A, P^B and P^C yield a solution for the Disjoint Double Digest instance: By construction, the distances in each point set yield exactly the corresponding set of distances. Each point in P^A is the sum of an integer less than t and an even multiple of t. On the other hand, each point in P^B except for the first and the last one is the sum of a

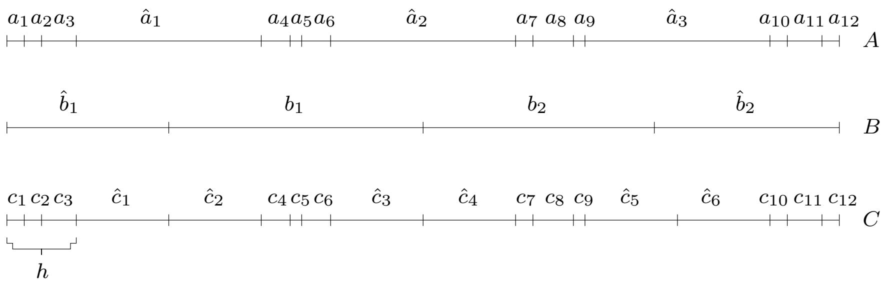


Figure 3.1: Solution for Disjoint Double Digest, for $n = 4$.

multiple of h and an odd multiple of t. Thus, sets P^A and P^B are disjoint except for the first and the last point. Moreover, $P^C = P^A \cup P^B$, hence the three point sets yield a solution for the Disjoint Double Digest instance.

For the opposite direction, let P^A, P^B and P^C be a solution for the Disjoint Double Digest instance. We consider only sets P^B and P^C. Each of the $n+1$ points in P^B consists of a multiple of h and a multiple of t, and each two points in P^B differ in the multiplicity of h. Since $P^B \subseteq P^C$, there must exist $n+1$ points in C that are of the same form. Each point in P^C corresponds to the sum of some distances from C. The distances $\hat{c}_j$ contribute only to the multiplicity of t by construction. Thus, the points in P^C must be such that the distances c_i "generate" the $n+1$ points with different multiples of h. Since 0 is the minimal point in C, this yields n subsets of c_i's that each sum up to h. Moreover, with $c_i = q_i$ and $\frac{h}{4} < q_i < \frac{h}{2}$ by definition, each of the n subsets has exactly three elements. Thus, the corresponding triples of q_i's are a solution for the 3–Partition instance.

□

3.3 Approximability of Optimization Variations of Double Digest

In this section, we study the approximability of optimization variations of Double Digest. In particular, we show that Min Absolute Error Double Digest cannot be approximated at all, while we give constant upper and lower bounds for the approximability of the Min Relative Error Double Digest and Min Point Number Double Digest.

First, we show that there is no polynomial time algorithm for Min Absolute Error Double Digest that achieves any finite approximation ratio.

Theorem 3.3.1. Min Absolute Error Double Digest *cannot be approximated to within any finite approximation ratio, unless* $\mathsf{P} = \mathsf{NP}$.

Proof: By contradiction, assume the existence of a polynomial–time approximation algorithm $\mathcal{A}$ with finite approximation ratio r. Then we have $e(\text{solution of algorithm } \mathcal{A} \text{ for } I) \leq r \cdot e(\text{any optimal solution for } I)$ for any instance I. This is also true for instances that actually have no partial cleavage error at all, and are thus instances of Double Digest. For such instances, an optimal solution has error 0, and therefore the approximation algorithm needs to find a solution with no error as well. Hence, this algorithm could be used to decide the Double Digest problem, which is in fact NP-complete [41].

□

We now show that the problem Min Point Number Double Digest is hard to approximate, by giving a gap–preserving reduction from Max 3–Dimensional Matching.

Theorem 3.3.2. Min Point Number Double Digest *cannot be approximated to within* $\frac{392}{391}$*, unless* $\mathsf{P} = \mathsf{NP}$.

Proof: Let T be a given instance of Max 3–Dimensional Matching with $T \subseteq W \times X \times Y$, where $W = \{w_1, \ldots, w_d\}, X = \{x_1, \ldots, x_d\}$ and $Y = \{y_1, \ldots, y_d\}$. Let $n = |T|$. We construct an instance of Min Point Number Double Digest as follows: Let base $Z = d^2 + 1$. Let $w'_i = \Delta_{3d}(i)$, $x'_i = \Delta_{3d}(d + i)$, and $y'_i = \Delta_{3d}(2d + i)$ for $1 \leq i \leq d$. Each of the digits in these numbers corresponds one–to–one to a value from $W \cup X \cup Y$, and the sum over all elements w'_i, x'_i and y'_i is $\mathbb{1}_{3d}$. For each triple $t_l = (w_i, x_j, y_k) \in T$ we define $t'_l = w'_i + x'_j + y'_k$. Moreover, let $z = \sum_{t_l \in T} t'_l - \mathbb{1}_{3d}$. We define multiset A containing all numbers w'_i, x'_j, y'_k, and number z; multiset B contains all values t'_l; and C is the same as A. Then $\operatorname{sum}(A) = \operatorname{sum}(B) = \operatorname{sum}(C)$ (due to the choice of z), and the three multisets are a valid instance of Min Point Number Double Digest.

We denote solutions of the Max 3–Dimensional Matching instance by SOL, and solutions of the Min Point Number Double Digest instance by SOL'. We now show the following equivalence:

$$\exists SOL : |SOL| \geq m \iff \exists SOL' : |SOL'| \leq 3d + n + 1 - m. \qquad (3.1)$$

To prove the direction from left to right in (3.1), let SOL be a solution of the Max 3–Dimensional Matching instance with at least m triples. W.l.o.g.

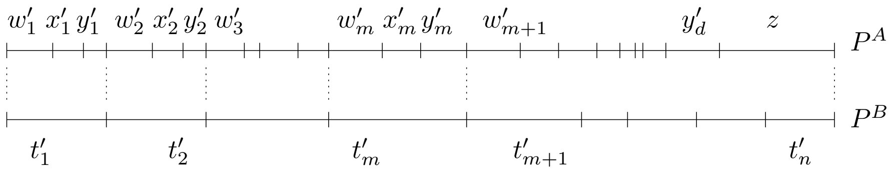

w'_1 x'_1 y'_1 w'_2 x'_2 y'_2 w'_3
w'_m x'_m y'_m w'_{m+1}
y'_d
z
P^A
P^B
t'_1
t'_2
t'_m
t'_{m+1}
t'_n

Figure 3.2: Solution SOL' for MIN POINT NUMBER DOUBLE DIGEST instance.

assume that $t_1, \dots, t_m$ are the triples in SOL, and that $t_l = (w_l, x_l, y_l)$ for $1 \le l \le m$. This assumption is valid since each element from $W \cup X \cup Y$ occurs at most once in the first m triples. We define a solution SOL' for the MIN POINT NUMBER DOUBLE DIGEST instance as follows (cf. Figure 3.2): Starting in 0, we define $n+1$ points P^B on a line such that the adjacent distances between the first $m+1$ points are exactly values $t'_1, \dots, t'_m$, and the distances between the other points are the remaining values from B. Analogously, we define $3d+2$ points P^A such that the distances of adjacent points are, in that ordering, $w'_1, x'_1, y'_1, w'_2, x'_2, y'_2, w'_3, \dots, y'_d, z$. With $P^C = P^A$, the three point sets yield a solution for the MIN POINT NUMBER DOUBLE DIGEST instance with at most $3d+n+1-m$ points. To see this, observe there are $3d+2$ points in P^A and $n+1$ points in P^B. Since $P^C = P^A$, set P^C does not contribute to the total number of points. All points in P^B corresponding to the first m triples in B occur in P^A as well. In addition, the two sets agree in 0 and the last point. Hence, the number of points in $P^A \cup P^B$ is at most $3d+2+n+1-(m+2)$.

For the direction from right to left in (3.1), let $SOL' = (P^A, P^B, P^C)$ be a solution of the MIN POINT NUMBER DOUBLE DIGEST instance with at most $3d+n+1-m$ points. If $P^C \neq P^A$ in SOL', we can construct a new solution by setting $P^C = P^A$. This does not increase the number of points, hence, we assume in the following that $P^C = P^A$. There exist two points $p, q \in P^A \cap P^B$ such that $p < q$, such that z is the distance between some two points between p and q, and such that no other point from $P^A \cap P^B$ is inbetween p and q (cf. Figure 3.3). We assume in the following that $q = \text{sum}(A)$, i.e., z is between the two rightmost points from $P^A \cap P^B$. If this is not the case in SOL', we can achieve this by swapping the block between p and q and the block between q and $\text{sum}(A)$, for each set P^A, P^B, and P^C. This swap operation does not change the total number of points.

We now define a solution for the MAX 3–DIMENSIONAL MATCHING instance: Let $SOL \subseteq T$ be the set of all triples that correspond to a value in B that is the distance between two adjacent points in P^B that are to the left of p. We now show that the triples in SOL are disjoint. Since p is a

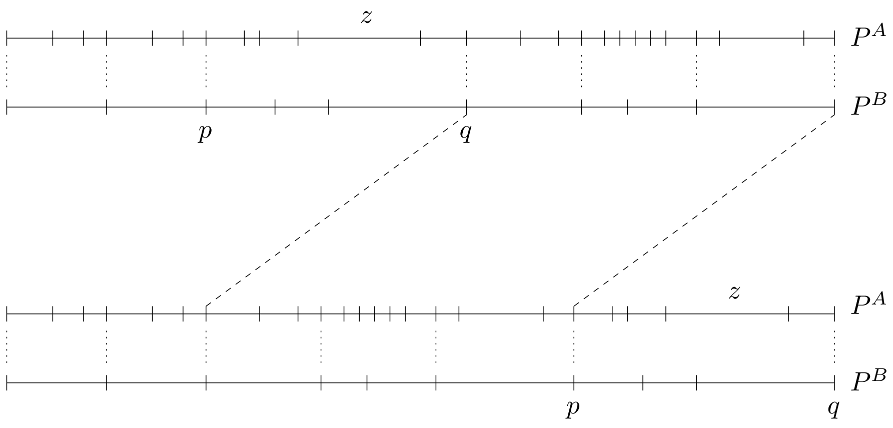


Figure 3.3: Swap operation moving the block that contains distance z to the end of P^A.

point in both P^A and P^B, the sums of the distances from A and from B, respectively, that occur to the left of p are equal. Let s be this sum. Then s is a number with $3d$ digits. Each digit in s is either 0 or 1, since no two values in $A - \{z\}$ (recall that z is *not* to the left of p) have a one in the same digit. Hence, no two values from B that are to the left of p can have a one in the same digit, since we chose base Z sufficiently large such that no carry–overs occur. Since each digit corresponds one–to–one to an element from $W \cup X \cup Y$, the triples in SOL must be disjoint. Moreover, each such element from $W \cup X \cup Y$ occurs in exactly one triple from SOL, i.e., SOL is a perfect matching for this set of elements. In the following, we assume that the values from A and B that are to the left of p are arranged such that the three elements that belong to a single triple are adjacent (analogous to Figure 3.2). If this is not the case in SOL', we can rearrange the points in P^A accordingly without increasing the total number of points. Observe that this assumption ensures that between two adjacent matching points from P^A and P^B that are to the left of p we have exactly one value from B (and three values from A, respectively).

By assumption, the total number of points in SOL' is at most $3d + n + 1 - m$. Point set P^A must contain $3d + 2$ points (since $|A| = 3d + 1$) and point set P^B contains $n + 1$ points. Hence, there are at least $m + 2$ points that occur in both P^A and P^B. By definition of point p, there exists only one point to the right of p that is in both P^A and P^B, namely q. Hence, at least $m + 1$ points to the left of p, inclusive, occur in both P^A and P^B. Thus, there are at least m triples in SOL, each of them corresponding to one of the values from B that is to the left of p. This finishes our proof of Equivalence (3.1).

We now show how a gap–problem of MAX 3–DIMENSIONAL MATCHING transforms into a gap–problem of MIN POINT NUMBER DOUBLE DIGEST. Let OPT and OPT' denote the size of optimum solutions for the MAX 3–DIMENSIONAL MATCHING and the MIN POINT NUMBER DOUBLE DIGEST instance, respectively. For constants $\alpha, \beta > 0$, Equivalence (3.1) yields the following implications:

- $OPT \geq (1-\alpha)d \Rightarrow OPT' \leq 3d+n+1-(1-\alpha)d$
- $OPT < (1-\beta)d \Rightarrow OPT' > 3d+n+1-(1-\beta)d$

Figuratively, this means the following: Given a gap–problem of MAX 3–DIMENSIONAL MATCHING, our reduction transforms it into a gap–problem of MIN POINT NUMBER DOUBLE DIGEST such that the width of the gap remains the same, but it is "reflected". Observe that the range of possible solutions increases from d triples to $3d+n+1$ points. Furthermore, observe that applying Equivalence (3.1) to optimum solutions yields $OPT = m \iff OPT' = 3d+n+1-m$.

To finish our proof, we now show that MIN POINT NUMBER DOUBLE DIGEST is hard to approximate. It is NP-hard to decide for MAX 3–DIMENSIONAL MATCHING whether $OPT \geq (1-2\delta+\varepsilon)d$, or $OPT < (1-3\delta)d$, for any constant $0 < \delta \leq \frac{1}{97}$ and arbitrarily small $\varepsilon > 0$ [15, 16, 17]. This result even holds for the restricted version of the problem where every element from $W \cup X \cup Y$ occurs in exactly 2 triples. In this case $n = 2d$. Using our reduction and the two implications above, we have shown that it is NP-hard to decide for MIN POINT NUMBER DOUBLE DIGEST whether $OPT' \leq 4d+1+(2\delta-\varepsilon)d$ or $OPT' > 4d+1+3\delta d$, for any constant $0 < \delta \leq \frac{1}{97}$ and arbitrarily small $\varepsilon > 0$. With $\delta = \frac{1}{97}$ and sufficiently large instances (i.e., $d > 97$), we have

$$\begin{aligned}
\frac{4d+1+3\delta d}{4d+1+(2\delta-\varepsilon)d} &= 1+\frac{\delta d+\varepsilon d}{4d+1+(2\delta-\varepsilon)d} \\
&> 1+\frac{\delta d}{4d+1+2\delta d} \\
&= 1+\frac{\delta}{4+\frac{1}{d}+2\delta} \\
&> 1+\frac{\delta}{4+3\delta} \\
&= 1+\frac{1}{391}.
\end{aligned}$$

Thus, Min Point Number Double Digest cannot be approximated to within $1 + \frac{1}{391}$, unless $\mathsf{P} = \mathsf{NP}$.

$\square$

Observe that Min Point Number Double Digest can be approximated with factor 3: If all distances from an instance A, B and C are arranged on three lines in an arbitrary fashion, each starting in 0, then this results in a solution with at most $|A| + |B| + |C| - 1$ points; on the other hand, an optimum solution will always use at least $\max(|A|, |B|, |C|) + 1$ points. Thus, this trivial "algorithm" achieves an approximation ratio of 3 for Min Point Number Double Digest.

Similarly, the problem Min Relative Error Double Digest can be approximated with factor 2: We can again arrange all distances from an instance A, B and C in an arbitrary fashion. Then we obtain a solution with an optimization measure of at most $r(P^A, P^B, P^C) = |A| + |B| + |C| + |A| + |B| + |C| - 3$, since not a single point might be matched except for the first and the last point. In an optimum solution, the optimization measure would be at least $|A| + |B| + |C|$, thus giving an approximation ratio of 2 for this "algorithm".

We now use basically the same proof technique as for the previous theorem to prove that Min Relative Error Double Digest is hard to approximate.

Theorem 3.3.3. Min Relative Error Double Digest *cannot be approximated to within* $\frac{877}{876}$, *unless* $\mathsf{P} = \mathsf{NP}$.

Proof: We use the same reduction as in Theorem 3.3.2 and show the following equivalence (recall that $r(SOL')$ is defined as $|A| + |B| + |C| + |(P^A \cup P^B) - P^C| + |P^C - (P^A \cup P^B)|$):

$$\exists SOL : |SOL| \geq m \iff \exists SOL' : r(SOL') \leq 6d + 2n + 1 - m. \qquad (3.2)$$

The implication from left to right can be shown by using the same arguments as in the proof of Theorem 3.3.2: Given a solution SOL with at least m triples, we define a solution SOL' like in the previous proof. With $P^A = P^C$ by construction, we have $r(SOL') = |A| + |B| + |C| + |P^B - P^C| \leq (3d + 1) + n + (3d + 1) + n - m - 1 = 6d + 2n + 1 - m$.

For the opposite direction of the equivalence, let $SOL' = (P^A, P^B, P^C)$ be a solution with $r(SOL') \leq 6d + 2n + 1 - m$. If $P^A \neq P^C$, then the number of unmatched points does not increase by setting $\hat{P}^A = P^C$, since in this case $|(\hat{P}^A \cup P^B) - P^C| = |P^B - P^C| \leq |(P^A \cup P^B) - P^C|$ and $|P^C - (\hat{P}^A \cup P^B)| = 0 \leq |P^C - (P^A \cup P^B)|$. Hence, constructing a solution

SOL from *SOL'* as in the proof of Theorem 3.3.2, and using the same arguments, shows Equivalence 3.2.

Using Equivalence 3.2 and the fact that Max 3–Dimensional Matching is hard to approximate [15, 16, 17], the claim is shown analogous to the proof of Theorem 3.3.2. □

3.4 NP-hardness of Finding Feasible Solutions for Optimization Variations of Disjoint Double Digest

In this section, we show that no Disjoint Double Digest optimization variation can be approximated by any polynomial–time approximation algorithm with a finite approximation ratio, unless P = NP. We achieve this by showing that even finding feasible solutions for such problems is NP-hard. To this end, we first show that the problem Disjoint Ordering (see Definition 1.4.1) is NP-complete. We then show how to reduce Disjoint Ordering to any optimization variation of Disjoint Double Digest.

Lemma 3.4.1. Disjoint Ordering *is* NP*-complete.*

Proof: Obviously, Disjoint Ordering is in NP. To show NP-hardness, we reduce 3–Partition to it. Given an instance $q_1, \ldots, q_{3n}$ and h of 3–Partition, we construct an instance of Disjoint Ordering as follows. Let

$$\begin{aligned} a_i &= q_i && \text{for } 1 \le i \le 3n, \\ \hat{a}_j &= h && \text{for } 1 \le j \le n+1, \\ b_i &= h+2 && \text{for } 1 \le i \le n, \text{ and} \\ \hat{b}_j &= 1 && \text{for } 1 \le j \le (n+1) \cdot h - 2n. \end{aligned}$$

Let A consist of the a_i's and $\hat{a}_j$'s, and let B consist of the b_i's and $\hat{b}_j$'s. Then $\text{sum}\,(A) = \text{sum}\,(B) = (2n+1) \cdot h$. The number of distances in A is polynomial in n, while the cardinality of B is only polynomial in n and h. However, since 3–Partition is NP-complete in the strong sense, it is still NP-complete if h is polynomially bounded in n. In this case, A and B are an instance of Disjoint Ordering which can be constructed in time polynomial in n. We now show that any solution for the 3–Partition

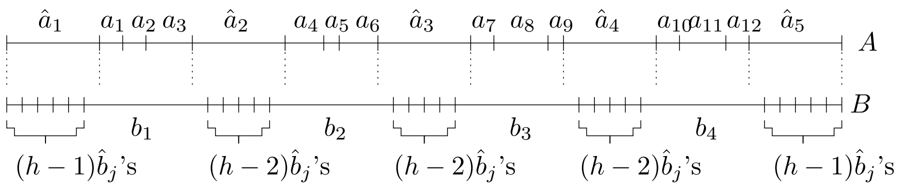


Figure 3.4: Disjoint ordering of distances in A and B, for $n = 4$. Dotted lines have distance h.

instance yields a solution for the Disjoint Ordering instance, and vice versa.

If there is a solution for the 3–Partition instance, then there exist n disjoint triples of q_i's such that each triple sums up to h. W.l.o.g., we assume that the q_i's are ordered such that each three q_i's from one triple are adjacent. We arrange the distances from A on a line, starting in 0, such that each three a_i's that belong to the same triple are adjacent, and such that each three a_i's are separated by one $\hat{a}_j$ (see Figure 3.4). The distances from B are arranged on a line as follows: First we have $h-1$ distances $\hat{b}_j$, followed by n combinations of one distance b_i and $h-2$ distances $\hat{b}_j$, and at the end there are again $h-1$ distances $\hat{b}_j$. Let P^A and P^B be the corresponding point sets. Then P^A and P^B are disjoint except for the first and the last point, and they yield a solution for the Disjoint Ordering instance.

For the opposite direction, assume that P^A and P^B are a solution for the Disjoint Ordering instance. We first show that the ordering of the distances from B constructed in the previous paragraph is the only possible arrangement. In P^B, there are n distances b_i. They separate at most $n+1$ blocks of consecutive distances $\hat{b}_j$, including the two margin blocks. Some of the blocks might be empty. Since h is the largest number in A, the length of a margin block is at most $h-1$, and the length of an inner block is at most $h-2$. Thus, the total length of the blocks of $\hat{b}_j$'s is at most $2 \cdot (h-1) + (n-1) \cdot (h-2) = (n+1)h - 2n$. This is exactly the number of distances $\hat{b}_j$, and therefore their total length. Thus, each of the previous upper bounds has to be tight. This yields the ordering of the distances from B presented above (see Figure 3.4). For the ordering in P^A, the $n+1$ distances $\hat{a}_j$ must be used to cover the $n+1$ blocks of consequtive $\hat{b}_j$. This leaves exactly n gaps, each of length h, which are covered by the distances a_i. This yields a solution for the 3–Partition instance, since $\frac{h}{4} < q_i < \frac{h}{2}$ by definition, which implies that each gap is covered by exactly three distances.

□

We now show how to reduce Disjoint Ordering to Min Relative Error Disjoint Double Digest: Let A and B be an instance of Disjoint Ordering. We "construct" an instance of Min Relative Error Disjoint Double Digest by simply letting multisets A and B be the same, and multiset C be the empty set. If an approximation algorithm for Min Relative Error Disjoint Double Digest finds a feasible solution for this instance, this yields immediately a solution for the Disjoint Ordering instance, since any feasible solution for Min Relative Error Disjoint Double Digest must arrange the elements from A and B in a disjoint fashion.

The same argument applies to Min Point Number Disjoint Double Digest, and for any other (reasonable) optimization variation of Disjoint Double Digest, since the reduction is totally independent of the optimization criterion. Thus, we have the following.

Theorem 3.4.2. *No polynomial–time approximation algorithm can achieve a finite approximation ratio for any (reasonable) optimization variation of* Disjoint Double Digest*, unless* $\mathsf{P} = \mathsf{NP}$.

3.5 Conclusion

We have shown that Double Digest and Disjoint Double Digest are NP-complete in the strong sense. Moreover, we studied the approximability of three optimization variations of Double Digest that model partial cleavage errors. We proved that Min Absolute Error Double Digest cannot be approximated by any finite approximation ratio, and showed that the problems Min Relative Error Double Digest and Min Point Number Double Digest cannot be approximated to within factor $\frac{877}{876}$ and $\frac{392}{391}$, respectively, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, arranging the distances in an arbitrary fashion yields already solutions for both problems that are only a factor 2 and 3, respectively, off the optimum. In a last set of results, we showed for Disjoint Double Digest optimization variations that even finding feasible solutions is NP-hard. To this end, we proved that the problem Disjoint Ordering is NP-complete.

While our approximability results are tight for all Disjoint Double Digest variations, our results leave considerable gaps regarding the exact approximability threshold for Min Relative Error Double Digest and Min Point Number Double Digest, which present challenges for future research.

Moreover, optimization variations of Double Digest that model other error types (e.g. wrong fragment lengths or additional fragments) or even combinations of different error types should be defined and studied. On a meta–level of arguing, it seems unlikely that an optimization variation that models partial cleavage errors *and* some other error types could be any easier than the problems that model only partial cleavage errors, but there is a possibility that some error types might offset each other in a cleverly defined optimization problem.

Chapter 4

Partial Digestion

4.1 Introduction

In the Partial Digest problem, we are given a multiset D of distances and we ask for a set P of points on a line such that D is the pairwise distance multiset for P. We recapitulate the definition from the introduction (cf. Definition 1.2.2):

Definition. *Given an integer m and a multiset D of $k = \binom{m}{2}$ positive integers, is there a set $P = \{p_1, \ldots, p_m\}$ of m points on a line such that $\{|p_i - p_j| \mid 1 \leq i < j \leq m\} = D$?*

The exact computational complexity of Partial Digest is a long–standing open problem, and it appears in its pure combinatorial formulation already in the 1930's in the area of X–ray crystallography (acc. to [81]). The problem can be solved in pseudo–polynomial time [56, 75], and there exists a backtracking algorithm (for exact or erroneous data) that has expected running time polynomial in the number of distances [81, 82], but exponential worst case running time [100]. The Partial Digest problem can be formalized by cut grammars, which have one additional symbol δ, the *cut*, that is neither a non–terminal nor a terminal symbol [78], and the problem is closely related to the theory of homometric sets[1] [81]. Finally, if the points in a solution do not have to be on a line, but only in d–dimensional space, then the problem is NP-hard [81]. However, for the original Partial Digest problem, neither a polynomial–time algorithm nor a proof of NP-hardness is known [10, 28, 64, 74, 71, 79].

[1] Two (non–congruent) sets of points are homometric if they generate the same multiset of pairwise distances.

In the biological setting of partial digestion, many experimental variations have been studied: Probed partial digestion, where probes (markers) are hybridized to partially digested DNA [1, 63]; simplified partial digestion, where clones are cleaved either in one or in all restriction sites [10]; labeled partial digestion, where both ends of the DNA molecule are labeled before digestion[2] [64]; and multiple complete digestion, where many different enzymes are used [34] (which is as well a generalization of double digestion). For an introduction to the PARTIAL DIGEST problem, see for instance the survey by Lemke et al. [55], and the books by Pevzner [71] or by Setubal and Meidanis [79].

In reality, the partial digest experiment cannot be conducted under ideal conditions, and thus errors occur in the data. In fact, there are four types of errors that occur in partial digest experiments [31, 39, 48, 82, 95]:

Additional fragments An enzyme may erroneously cut in some cases at a site that is similar, but not exactly equivalent to a restriction site; thus, some distances will be added to the data even though they do not belong there. Furthermore, fragments can be added through contamination with biological material, such as DNA from unrelated sources.

Missing fragments Obviously, partial cleavage errors (see page 28) lead to missing fragments. Furthermore, fragments are not detected by gel electrophoresis if their amount is insufficient to be detected by common staining techniques. Finally, small fragments may remain undetected at all since they run off at the end of the gel.

Fragment length Using gel electrophoresis, it is almost impossible to determine the exact length of a fragment. Typical error ranges are between 2% and 7% of the fragment length.

Multiplicity detection Determining the proper multiplicity of a distance from the intensity of its spot in the gel is almost impossible in practice.

We define variations of PARTIAL DIGEST for the first three types of errors, and prove hardness results for each of these variations. Intuitively, the problem of modeling real–life instances – in which *all* error types can occur – is even harder than having only one error type.

The MIN PARTIAL DIGEST SUPERSET problem models the situation of omissions, where we are given data for which we know that some distances are missing, and we search for a set of points on a line such that the number

[2]Note that labeled partial digestion is connected to de novo peptide sequencing using MS/MS data; we will study this problem in Chapter 7.

of missing distances is minimum. This problem is formally defined as follows (recall that $\Delta(P)$ denotes the multiset of all distances between any two points in P).

Definition 4.1.1 (Min Partial Digest Superset). *Given a multiset D of k positive integers, find the minimum number m such that there is a set P of m points on a line with $D \subseteq \Delta(P)$.*

For example, if $D = \{2,\ 5,\ 7,\ 7,\ 9,\ 14,\ 23\}$, then the point set $P = \{0, 7, 9, 14, 23\}$ (as shown in Figure 1.3 on page 5) would be a minimum solution for the Min Partial Digest Superset instance D. On the other hand, if $D' = \{2,\ 7,\ 9,\ 9,\ 16\}$, then the points in P would still cover all distances from D', but there exist solutions with fewer points that cover D', e.g. point set $P' = \{0, 2, 9, 18\}$ (yielding distance multiset $\{2, 7, 9, 9, 16, 18\}$).

We show in Section 4.2 that computing an optimal solution for the Min Partial Digest Superset problem is NP-hard, by giving a reduction from Equal Sum Subsets. Our result provides in a sense an answer to the open problem 12.116 in the book by Pevzner [71], which asks for an algorithm to reconstruct a set of points, given a subset of their pairwise distances.

We can even strengthen our hardness result by considering the problem t–Partial Digest Superset, where we restrict the cardinality of a solution to at most t, for some parameter t that is specified as a fixed function in $|D|$, the cardinality of the input distance multiset:

Definition 4.1.2 (t–Partial Digest Superset). *Given a multiset D of positive integers, is there a set P of $m \le t$ integers such that $D \subseteq \Delta(P)$?*

We show that the t–Partial Digest Superset problem is NP-hard for *any* parameter $t = f(|D|) := |D|^{\frac{1}{2}+\varepsilon}$, for any $0 < \varepsilon < \frac{1}{2}$. This result is tight in a sense, since any solution (even for the original Partial Digest) must have at least cardinality $\Omega(|D|^{\frac{1}{2}})$.

In Section 4.3, we study the Max Partial Digest Subset problem, which models the situation of additions: We are given data in which some wrong distances were added, and we search for a set of points on a line such that they cover a maximum number of the given distances. A formal definition is as follows.

Definition 4.1.3 (Max Partial Digest Subset). *Given a multiset D of k positive integers, find the maximum number m such that there is a set P of m points on a line with $\Delta(P) \subseteq D$.*

We show that there is no polynomial–time algorithm for this problem that guarantees an approximation ratio of $|D|^{\frac{1}{2}-\varepsilon}$ for any $\varepsilon > 0$, unless

NP = ZPP.[3] To establish this result, we give a gap–preserving reduction from Max Clique. We also point to a trivial approximation algorithm for Max Partial Digest Subset that achieves a matching asymptotic approximation ratio. Thus, our inapproximability result is tight up to low–order terms.

Our two optimization variations of the Partial Digest problem allow the multiset of pairwise distances in a solution to be either a superset (i.e., to cover all given distances in D plus additional ones) or a subset (i.e., to contain only some of the distances in D) of the input set D. If a polynomial–time algorithm existed for either Min Partial Digest Superset or Max Partial Digest Subset, we could use this algorithm to solve the original Partial Digest problem as well: Any YES instance of Partial Digest is an instance of both optimization problems whose optimum is $\frac{1}{2}+\sqrt{\frac{1}{4}+2k}$; any NO instance of Partial Digest is an instance of Max Partial Digest Subset (resp., Min Partial Digest Superset) whose optimum is at most $\frac{1}{2}+\sqrt{\frac{1}{4}+2k}-1$ (at least $\frac{1}{2}+\sqrt{\frac{1}{4}+2k}+1$, respectively).

As a third type of error that can occur in real–life data, we study measurement errors in Section 4.4. Algorithms for Partial Digest with inaccurate data have been studied intensively in the literature [31, 48, 82, 95], and different error models have been proposed, e.g. for measurement errors that are logarithmic in the size of the fragment length [90, 91, 95] or for intervals of absolute errors [2, 82].

The Partial Digest problem is known to be strongly NP-hard if additive error bounds that can be even zero can be assigned to each distance *individually* [55, 81]. However, this does not model reality appropriately, since in real–life data we cannot assume that even one single fragment length can be measured exactly, and moreover, we cannot expect individual error bounds. Therefore, we study the computational complexity of the variation of Partial Digest where *all* measurements are prone to *the same additive non–zero* error.

We say that value v matches a distance d up to (additive) error ε if $|v-d| \leq \varepsilon$; moreover, a multiset D is a distance multiset for point set P up to error ε, if each distance between any two points in P can be matched with a distance in D up to error ε, and this matching is bijective. The Partial Digest With Errors problem is defined as follows.

Definition 4.1.4 (Partial Digest With Errors)**.** *Given an integer m, a multiset D of $k=\binom{m}{2}$ positive integers, and an error bound $\varepsilon>0$, is there*

[3]A problem Π is in class ZPP if there is a probabilistic algorithm for Π with polynomial running time which never outputs a wrong result, and which fails with probability less than $\frac{1}{2}$.

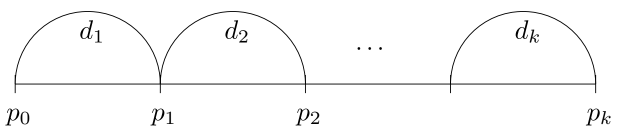


Figure 4.1: Trivial solution for a distance multiset D.

a set P of m points on a line such that D is the distance multiset for P up to error ε?

In Section 4.4, we prove that Partial Digest With Errors is strongly NP-complete by giving a reduction from 3–Partition.

Note that it would be even closer to real–life data to consider measurement errors that are *relative* to the distance length. We conjecture that the Partial Digest variation for relative errors is NP-hard as well, but we did not succeed to prove this conjecture. The same holds for the case of wrong multiplicities in the data.

Part of the results in this chapter have been published previously [21, 20].

4.2 NP-hardness of Min Partial Digest Superset

In this section, we study the Min Partial Digest Superset problem and show that this problem is NP-hard by giving a reduction from Equal Sum Subsets.

First observe that the minimum cardinality of a point set that covers all distances in a given multiset D cannot be too large. To see this, let $D = \{d_1, \ldots, d_k\}$ be a distance multiset. If m is the minimum number such that a set P of cardinality m with $D \subseteq \Delta(P)$ exists, then $m \leq k+1$: We set $p_0 = 0, p_i = p_{i-1} + d_i$ for $1 \leq i \leq k$, and $P_{triv} = \{p_0, \ldots, p_k\}$, i.e., we simply put all distances from D in a chain "one after the other" (see Figure 4.1). In P_{triv}, each distance d_i induces a new point, and we use one additional starting point 0. Obviously, set P_{triv} covers D and has cardinality $k+1$.

Observe that Partial Digest can be easily reduced to Min Partial Digest Superset: Given an instance D of Partial Digest of cardinality $|D| = k$, there is a solution for D if and only if the minimal solution for the Min Partial Digest Superset instance D has size $m = \frac{1}{2} + \sqrt{\frac{1}{4} + 2k}$ (in this case, $k = \binom{m}{2}$).

We now show that Min Partial Digest Superset is NP-hard by giving a reduction from Equal Sum Subsets (see Definition 1.4.2).

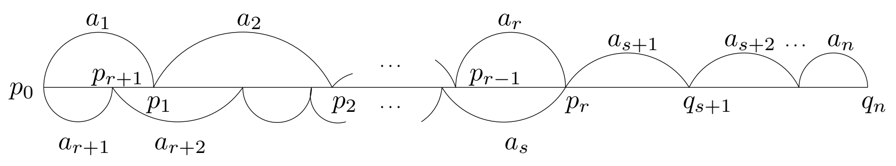


Figure 4.2: Solution if there are two sets of equal sum.

Theorem 4.2.1. Min Partial Digest Superset *is* NP*-hard.*

Proof: We reduce Equal Sum Subsets to Min Partial Digest Superset. Given an instance $A = \{a_1, \dots, a_n\}$ of Equal Sum Subsets, we set $D = A$ (and $k = n$), and prove in the following that there is a solution for the Equal Sum Subsets instance A if and only if a minimal solution for the Min Partial Digest Superset instance D has at most n points.

Let X and Y be a solution for the Equal Sum Subsets instance. Assume w.l.o.g. that $X = \{a_1, \dots, a_r\}$ and $Y = \{a_{r+1}, \dots, a_s\}$, for some $1 \le r < s \le n$. We construct a set P that covers D and that has at most cardinality n. Similarly to the construction of P_{triv}, we line up the distances from D. In this case, *two* chains start at point 0: Those distances from X and those from Y (see Figure 4.2); the remaining distances from $D-(X\cup Y)$ are positioned at the end of the two chains. More precisely, we set

$$\begin{aligned}
p_0 &= 0 \\
p_i &= p_{i-1} + a_i && \text{for } 1 \le i \le r \\
p_{r+1} &= a_{r+1} \\
p_j &= p_{j-1} + a_j && \text{for } r+2 \le j \le s-1 \\
q_{s+1} &= p_r + a_{s+1} \\
q_\ell &= q_{\ell-1} + a_\ell && \text{for } s+2 \le \ell \le n.
\end{aligned}$$

Set $P = \{p_0, \dots, p_{s-1}, q_{s+1}, \dots, q_n\}$ is the corresponding set of points. Notice that there is no point "p_s" in set P, since the two chains corresponding to X and Y share two points, namely $p_0 = 0$ and their common endpoint p_r.

Obviously, P is a set of cardinality n. Moreover, the definition of the points yields immediately that except for $i = s$ each a_i is the difference between two of the points (either $p_i - p_{i-1}$, or $q_{s+1} - p_r$, or $q_\ell - q_{\ell-1}$). To see that a_s occurs as well, first observe that $p_r = \sum_{i=1}^{r} a_i = \text{sum}\,(X)$ and that $p_{s-1} = \sum_{j=1}^{s-1} a_j = \text{sum}\,(Y) - a_s$. Thus, $p_r - p_{s-1} = \text{sum}\,(X) - (\text{sum}\,(Y) - a_s) = a_s$, since X and Y are a solution of the Equal Sum

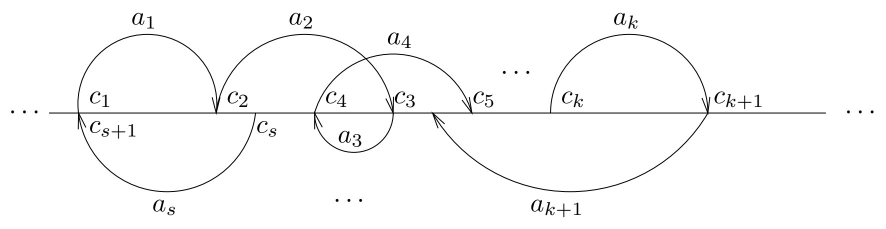


Figure 4.3: A solution containing a cycle yields two subsets of equal sum: the overall length of right jumps equals the overall length of left jumps.

Subsets instance and yield the same sum. Hence, P covers every distance from D.

For the opposite direction, let $P = \{p_1, \dots, p_m\}$ be an optimal solution for the Min Partial Digest Superset instance with $m < n + 1$. Since P covers D, for each $a \in D$ there is a pair (p, q) of points $p, q \in P$ such that $a = |p - q|$. For each $a \in D$ we choose one such pair and say that it is *associated* with value a. We define a graph $G = (V, E)$ with $V = P$ and $E = \{(p, q) \mid (p, q) \text{ is associated with some } a \in D\}$, i.e., G contains only those edges corresponding to some distance in D. Thus, $|V| = m$ and $|E| = |D| = n$. Since $m < n + 1$, this graph contains at least one cycle. We show in the following that such a cycle induces a solution for the Equal Sum Subsets instance.

Let $C = c_1, \dots, c_s$ be any cycle in G (see Figure 4.3). Then $|c_{i+1} - c_i| \in D$, for all $1 \le i \le s$ (here, we abuse notation and identify c_{s+1} with c_1). Assume w.l.o.g. that $|c_{i+1} - c_i|$ is associated with a_i, for $1 \le i \le s$. We define $I^+ := \{i \in \{1, \dots, s\} \mid c_{i+1} > c_i\}$, and $I^- := \{j \in \{1, \dots, s\} \mid c_{j+1} < c_j\}$, i.e., we partition the edges in the cycle into two sets, those that are oriented to the left (I^-) and those that are oriented to the right (I^+). This yields

$$\begin{aligned} 0 &= c_{s+1} - c_1 \\ &= \sum_{i=1}^{s} (c_{i+1} - c_i) \\ &= \sum_{i \in I^+} (c_{i+1} - c_i) + \sum_{j \in I^-} (c_{j+1} - c_j) \\ &= \sum_{i \in I^+} |c_{i+1} - c_i| - \sum_{j \in I^-} |c_{j+1} - c_j| \\ &= \sum_{i \in I^+} a_i - \sum_{j \in I^-} a_j. \end{aligned}$$

Sets $X := \{a_i \mid i \in I^+\}$ and $Y := \{a_j \mid j \in I^-\}$ yield equal sums, and thus a solution for the Equal Sum Subsets instance.

□

In the previous theorem, we have shown NP-hardness of Min Partial Digest Superset by reduction from Equal Sum Subsets. In the proof, we distinguished whether a minimal solution uses at most n points, or $n+1$ points (which in fact are always sufficient). We will now extend this result and allow to "decrease" the bound to some value t that is still sufficiently large. In fact, we show that the corresponding problem t–Partial Digest Superset is NP-hard for every $0 < \varepsilon < \frac{1}{2}$, if we set t to be $f(|D|) = |D|^{\frac{1}{2}+\varepsilon}$. Observe that for a distance multiset D, a minimal set of points covering D has cardinality at least $\frac{1}{2} + \sqrt{\frac{1}{4} + 2|D|} \approx |D|^{\frac{1}{2}}$. Moreover, the Partial Digest problem is equivalent to t–Partial Digest Superset with $t = \frac{1}{2} + \sqrt{\frac{1}{4} + 2|D|} = O\left(|D|^{\frac{1}{2}}\right)$.

Theorem 4.2.2. *t–Partial Digest Superset is NP-hard for any constant $0 < \varepsilon < \frac{1}{2}$ and for any $t = f(|D|) := |D|^{\frac{1}{2}+\varepsilon}$.*

Proof: We show NP-hardness by reduction from Equal Sum Subsets, analogous to the proof of Theorem 4.2.1. Let $\{a_1, \ldots, a_n\}$ be an instance of Equal Sum Subsets. Informally speaking, we "blow up" the instance of Min Partial Digest Superset used in the proof of Theorem 4.2.1 (cf. Figure 4.4): First, we have n distances in a set A', each corresponding to one of the a_i's. Then we add a set B of q "essential" distances (for some value q that we specify later) such that any solution for our instance must use exactly $q+1$ points to cover the distances in B, and no two of these points can be used to cover any distances from A'. Finally, we add a suitable set C' of $O(q^2)$ "inessential" distances to fill up the number of distances in our instance. Each distance in C' is the sum of some distances from B, and all the distances in C' can be covered "for free" by the points used for the distances in B (i.e., no additional points are necessary). Our instance D for t–Partial Digest Superset is the union of the distance sets A', B and C'. We will choose the size of set C' such that $t = f(|D|) = n + q$ holds. Moreover, we will show that either $n+q$ points are sufficient to cover all distances in our instance, or that we need at least $n+q+1$ points, and that there is a solution for the Equal Sum Subsets instance if and only if $n+q$ points are sufficient.

We postpone the choice of q and show first how the distance sets are defined. All distances are numbers with base $Z = q^2 + \sum_{i=1}^{n} a_i$ (recall that we represent large numbers as vectors; see Section 2.2). Let $a'_i = \langle a_i \rangle \circ \mathbb{0}_q$

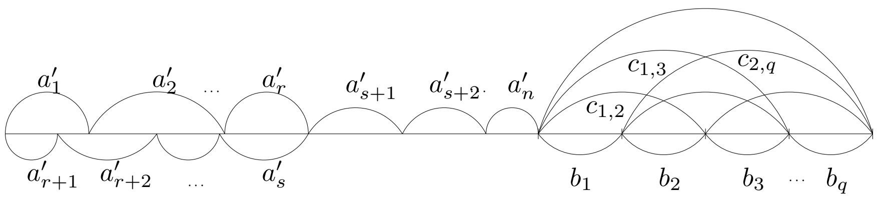


Figure 4.4: Distance sets A', B and C.

and $A' = \{a'_i \mid 1 \leq i \leq n\}$. For $1 \leq j \leq q$, let $b_j = \langle 0 \rangle \circ \Delta_q(j)$, and $B = \{b_j \mid 1 \leq j \leq q\}$. For $1 \leq u < v \leq q$, let $c_{u,v} = \sum_{\ell=u}^{v} b_\ell$, and $C = \{c_{u,v} \mid 1 \leq u < v \leq q\}$. Obviously, no distances from A' sum up to a distance in B or C, and vice versa.

The instance of t–Partial Digest Superset is defined by $D = A' \cup B \cup C'$, where C' is a subset of C of appropriate size. Clearly, $|D| = n + q + |C'|$. We want to choose the size of $|C'|$ such that $f(|D|) = (n+q+|C'|)^{\frac{1}{2}+\varepsilon} = n+q$ is satisfied. To this end, it suffices to take any $C' \subseteq C$ with cardinality $|C'| = (n+q)^{\frac{2}{1+2\varepsilon}} - (n+q)$. [If the latter number is not an integer, the proof can be easily adjusted by considering $|C'| = \lfloor (n+q)^{\frac{2}{1+2\varepsilon}} \rfloor - (n+q)$, and choosing q appropriately; this is possible for sufficiently large n.] In order to make this possible, we need to have $|C| \geq (n+q)^{\frac{2}{1+2\varepsilon}} - (n+q)$. Since C contains $\binom{q}{2}$ distances, we have to choose q sufficiently large to make the inequality $\binom{q}{2} \geq (n+q)^{\frac{2}{1+2\varepsilon}} - (n+q)$ hold. This inequality holds if we choose $q \geq \max\{6^{\frac{1}{\varepsilon}}, n\}$, which is shown as follows.

$$\begin{array}{llrr}
 & q \quad \geq & 6^{\frac{1}{\varepsilon}} & \text{(by assumption)} \\
\Rightarrow & q^{\frac{1}{2}} \quad \geq & 6^{\frac{1}{2\varepsilon}} & \\
\Rightarrow & q-2 \quad \geq & 6^{\frac{1}{2\varepsilon}} & (\text{since } q-2 > q^{\frac{1}{2}} \text{ for } q \geq 6^{\frac{1}{\varepsilon}} > 6) \\
\Rightarrow & (q-2)^{2\varepsilon} \quad \geq & 6 & \\
\Rightarrow & (q-2)^{2\varepsilon} \quad \geq & \dfrac{q}{q-2} \cdot 4 & (\text{since } \dfrac{3}{2} > \dfrac{q}{q-2} \text{ for } q > 6) \\
\Rightarrow & (q-2)^{2\varepsilon} \quad \geq & \dfrac{q}{q-2} \cdot 2^{\frac{3+2\varepsilon}{2}} & (\text{since } 4 > 2^{\frac{3+2\varepsilon}{2}} \text{ for } \varepsilon < \dfrac{1}{2})
\end{array}$$

$$\begin{array}{llll}
\Rightarrow & (q-2)^{1+2\varepsilon} & \geq & 2q \cdot \sqrt{2}^{1+2\varepsilon} \\
\Rightarrow & \left(\frac{q-2}{\sqrt{2}}\right)^{1+2\varepsilon} & \geq & n+q \qquad \text{(since } q \geq n\text{)} \\
\Rightarrow & \frac{(q-2)^2}{2} & \geq & (n+q)^{\frac{2}{1+2\varepsilon}} \\
\Rightarrow & \binom{q}{2} & \geq & (n+q)^{\frac{2}{1+2\varepsilon}}
\end{array}$$

We claim that there are two subsets of A of equal sum if and only if there is a set P of at most $t = n + q$ points such that $D \subseteq \Delta(P)$. The proof of this equivalence is based on the fact that, by construction, no subset of distances from $B \cup C'$ can have the same length as a subset of distances from A'. Therefore, we need $q+1$ points to cover all distances from $B \cup C'$. The remaining set A' behaves as in the proof of Theorem 4.2.1: By reusing one of the $q+1$ points, we need at most n further points to cover A'; as in the proof of Theorem 4.2.1, less than n points are sufficient if and only if there exists a solution for the Equal Sum Subsets instance.

□

4.3 Approximability of Max Partial Digest Subset

In this section, we show that Max Partial Digest Subset is as hard to approximate as Max Clique (see Definition 2.4.3), and we give a trivial approximation algorithm that achieves a matching approximation ratio.

In the following, we construct a gap-preserving reduction from Max Clique to Max Partial Digest Subset. The problem Max Clique cannot be approximated by any polynomial–time algorithm to within factor $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$, where n is the number of vertices of the input graph, unless $\mathsf{NP} = \mathsf{ZPP}$ [44, 62]. Our reduction is gap–preserving, thus the inapproximability of Max Clique is transferred to Max Partial Digest Subset.

Theorem 4.3.1. Max Partial Digest Subset *cannot be approximated to within factor* $|D|^{\frac{1}{2}-\varepsilon}$*, for any constant* $\varepsilon > 0$*, where* $|D|$ *is the number of input distances, unless* $\mathsf{NP} = \mathsf{ZPP}$.

Proof: Let $G = (V, E)$ be an instance of Max Clique with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subseteq V \times V$. We construct an instance D of Max Partial Digest Subset by creating a number $d_{i,j} = \mathbb{0}_i \circ \mathbb{1}_{j-i} \circ \mathbb{0}_{n-j}$ with base $Z = n^2 + 1$ for each $(v_i, v_j) \in E, j > i$.

Let OPT be the size of a maximum clique in G (i.e., the number of vertices in a maximum clique), let OPT' be the maximum number of points that can be placed on a line such that all pairwise distances appear in D, let $k > 0$ be an integer, and let $\varepsilon > 0$. We now prove the following two implications.

1. If $OPT \geq kn^{1-\varepsilon}$, then $OPT' \geq kn^{1-\varepsilon}$.
2. If $OPT < k$, then $OPT' < k$.

To see the first implication, assume we are given a clique in graph G of size $kn^{1-\varepsilon}$. We construct a solution for the corresponding Max Partial Digest Subset instance D by positioning a point at position $v'_i = \mathbb{1}_i \circ \mathbb{0}_{n-i}$ for each vertex v_i in the clique. This yields a feasible solution for D, since for $j > i$ each distance $v'_j - v'_i = \mathbb{0}_i \circ \mathbb{1}_{j-i} \circ \mathbb{0}_{n-j} = d_{i,j}$ between two points v'_j and v'_i corresponds to an edge in G and is therefore encoded as distance $d_{i,j}$ in D.

We now show the second implication by proving its converse, i.e., by showing $OPT' \geq k \implies OPT \geq k$. Suppose we are given a solution of the Max Partial Digest Subset instance consisting of k points $p_1 < \ldots < p_k$ on a line. We assume w.l.o.g. that $p_1 = \mathbb{0}_n$. Let $d_{i_{\min}, j_{\max}} = p_k - p_1$. Note that $d_{i_{\min}, j_{\max}}$, and thus $i_{\min}$ and $j_{\max}$, are uniquely defined by construction. Each of the points $p_2, \ldots, p_{k-1}$ from the solution has the following properties:

1. It only has zeros and ones in its digits, as the distance to point p_1 would not be in D otherwise.
2. It only has zeros in the first $i_{\min}$ digits, as the distance to point p_k would not be in D otherwise.
3. It contains at most one continuous block of ones in its digits, as the distance to point p_1 would not be in D otherwise.

The points $p_2, \ldots, p_{k-1}$ also have the property that they are of the same form,

$$\begin{array}{lccccccc} \text{either} & \mathbb{0}_{i_{\min}} & \circ & \mathbb{1}_{\ell} & \circ & \mathbb{0}_{j_{\max}-\ell-i_{\min}} & \circ & \mathbb{0}_{n-j_{\max}} \\ \text{or} & \mathbb{0}_{i_{\min}} & \circ & \mathbb{0}_{\ell} & \circ & \mathbb{1}_{j_{\max}-\ell-i_{\min}} & \circ & \mathbb{0}_{n-j_{\max}} \end{array}$$

where $0 \leq \ell \leq j_{\max} - i_{\min}$. Only one of the two forms can occur in a solution, since if both forms existed, i.e., at least one point of each form existed, then the distance between points of different form would not be in D, since at least one digit would not be 0 or 1.

We now construct a vertex set V' that will turn out to be a clique. Let $v_{i_{\min}}$ and $v_{j_{\max}}$ be in vertex set V'. In addition, for each point $p_{k'}$, $2 \leq k' \leq k-1$, we have one vertex in set V': If $p_{k'}$ is of the first form, i.e., $p_{k'} = \mathbb{0}_{i_{\min}} \circ \mathbb{1}_{\ell'} \circ \mathbb{0}_{j_{\max}-\ell'-i_{\min}} \circ \mathbb{0}_{n-j_{\max}}$ for some $\ell' \in \{0, \ldots, j_{\max} - i_{\min}\}$, then we include $v_{\ell'+i_{\min}}$. Analogously, if $p_{k'}$ is of the second form, i.e., $p_{k'} = \mathbb{0}_{i_{\min}} \circ \mathbb{0}_{\ell'} \circ \mathbb{1}_{j_{\max}-\ell'-i_{\min}} \circ \mathbb{0}_{n-j_{\max}}$ for some $\ell' \in \{0, \ldots, j_{\max} - i_{\min}\}$, then we include $v_{\ell'+i_{\min}}$.

In order to see that the vertex set V' is a clique, consider the difference $p_{k'} - p_{k''}$ of any two points with $k' > k''$, where $p_{k'}$ has led to the inclusion of vertex $v_{\ell'}$ into the set and $p_{k''}$ has led to the inclusion of vertex $v_{\ell''}$ into the clique. This difference is exactly $d_{\ell',\ell''}$, and thus the edge $(v_{\ell'}, v_{\ell''})$ is in E.

The promise problem of Max Clique, in which we are promised that the size of the maximum clique in a given graph G is either at least $kn^{1-\varepsilon}$, or less than k, and we are to decide which is true, is hard to decide [44]. The two implications above show that our reduction transforms this promise problem of Max Clique into a promise problem of Max Partial Digest Subset, in which we are promised that in an optimum solution of D either at least $kn^{1-\varepsilon}$, or less than k points can be placed on a line. This promise problem of Max Partial Digest Subset is hard to decide as well, since a polynomial–time algorithm for it could be used to decide the promise problem of Max Clique. Thus, unless $\mathsf{NP} = \mathsf{ZPP}$, Max Partial Digest Subset cannot be approximated with an approximation ratio of

$$\frac{kn^{1-\varepsilon}}{k} = n^{1-\varepsilon} \geq |D|^{\frac{1}{2}-\varepsilon},$$

where $|D|$ is the number of distances in instance D. This yields the claim.

□

A trivial approximation algorithm for Max Partial Digest Subset works as follows: Given an instance $D = \{d_1, \ldots, d_{|D|}\}$, it simply places two points at distance d_1 from each other. This approximation algorithm achieves an approximation ratio of $O(|D|^{\frac{1}{2}})$, since any optimal solution has at most $\frac{1}{2} + \sqrt{\frac{1}{4} + 2|D|}$ points. This matches our lower bound up to lower order terms.

4.4 Strong NP-completeness of Partial Digest With Errors

In this section, we prove that PARTIAL DIGEST WITH ERRORS is strongly NP-complete by giving a reduction from 3–PARTITION (see Definition 2.4.1).

The idea of the reduction is as follows. Given an instance $q_1, \ldots, q_{3n}$ and h of 3–PARTITION, we define a multiset of distances D and an error $\varepsilon = \frac{h}{4}$ that form an instance of PARTIAL DIGEST WITH ERRORS. Our construction is based on the following observation: If there is a solution for the 3–PARTITION instance, then we can arrange the q_i's such that triples of adjacent q_i's sum up to h. If we sum up, say, 25 adjacent q_i, then we sum over at least 7 complete triples (that have sum h), plus some few (up to four) additional q_i's at the beginning and the end. In the special and trivial case that all q_i's have exactly value $\frac{h}{3}$, we can easily determine the exact sum of the 25 values. However, in a given instance of 3–PARTITION typically not all q_i's will have value $\frac{h}{3}$. However, they have "approximately" value $\frac{h}{3}$, since they satisfy $\frac{h}{4} < q_i < \frac{h}{2}$ by definition. In the proof of the following theorem, we will use error ε to "close the gap" between $\frac{h}{3}$ and the true values of the q_i's.

Theorem 4.4.1. PARTIAL DIGEST WITH ERRORS *is strongly* NP*-complete.*

Proof: The problem PARTIAL DIGEST WITH ERRORS is obviously in NP. To prove strong NP-hardness, we give a reduction from 3–PARTITION. Given an instance of 3–PARTITION, i.e., integers $q_1, \ldots, q_{3n}$ and integer h, we define a distance multiset D and an error ε that are an instance of PARTIAL DIGEST WITH ERRORS. There will be a solution for this instance if and only if there is a solution for the 3–PARTITION instance. Parallel to the definition of D, we show already the "if" direction of the previous statement: To this end, we assume that the 3–PARTITION can be solved, i.e., there are n triples $T_1, \ldots, T_n$ of q_i's that each sum up to h, and we show how to construct a point set P that is a solution for the PARTIAL DIGEST WITH ERRORS instance, i.e., P matches D up to error ε. The opposite direction ("only if") is shown in a second step. We want to stress at this point that although the definition of D and the construction of P are presented simultaneously, the definition of D itself does *not* rely on the fact that there exists a solution for the 3–PARTITION instance.

We assume that $\frac{h}{12}$ is integer. [Otherwise, we can achieve this by simply multiplying all values q_i and h by 12.] Moreover, we assume w.l.o.g. that the values $q_1, \ldots, q_{3n}$ are ordered such that the three q_i's that belong to the same triple T_j are adjacent, i.e., $T_1 = (q_1, q_2, q_3), T_2 = (q_4, q_5, q_6)$, and so on. Finally, we assume that the elements in each T_i are sorted in ascending order, i.e., $q_1 \leq q_2 \leq q_3, q_4 \leq q_5 \leq q_6$, and so on. This ordering allows us to derive a set of inequalities for the $q_i's$. Let $(q_{3k+1}, q_{3k+2}, q_{3k+3})$ be a triple that sums up to h, for $0 \leq k \leq n-1$. Then $q_{3k+1} \leq \frac{h}{3}$, since q_{3k+1} is the smallest of the three elements in the triple, and not all of them can be greater than $\frac{h}{3}$. Similarly, $\frac{h}{3} \leq q_{3k+3}$. With $q_{3k+1} + q_{3k+2} = h - q_{3k+3}$, we have $q_{3k+1} + q_{3k+2} \leq h - \frac{h}{3} = \frac{2h}{3}$. In combination with the restriction $\frac{h}{4} < q_i < \frac{h}{2}$ (from the definition of 3–Partition), this yields the following inequalities:

$$\begin{array}{rll}
\frac{h}{4} < & q_{3k+1} & \leq \frac{h}{3} \\
\frac{h}{4} < & q_{3k+2} & < \frac{h}{2} \\
\frac{h}{3} \leq & q_{3k+3} & < \frac{h}{2} \\
\frac{h}{2} < & q_{3k+1} + q_{3k+2} & \leq \frac{2h}{3} \\
\frac{2h}{3} \leq & q_{3k+2} + q_{3k+3} & < h \\
h = & q_{3k+1} + q_{3k+2} + q_{3k+3} &
\end{array} \tag{4.1}$$

Equivalently, we can express these inequalities using $H := \frac{h}{12}$:

$$\begin{array}{rll}
3H < & q_{3k+1} & \leq 4H \\
3H < & q_{3k+2} & < 6H \\
4H \leq & q_{3k+3} & < 6H \\
6H < & q_{3k+1} + q_{3k+2} & \leq 8H \\
8H \leq & q_{3k+2} + q_{3k+3} & < 12H \\
12H = & q_{3k+1} + q_{3k+2} + q_{3k+3} &
\end{array} \tag{4.2}$$

We will use these inequalities later to derive upper and lower bounds for the error that we need to apply to our distances in order to guarantee the existence of a solution for the Partial Digest With Errors instance.

Before we define our distances, we need to introduce the *level* of a distance: For a point set P, we say that a distance d between two points has *level* ℓ if it spans $\ell - 1$ further points, and we say that distance d is an *atom* if it has level 1 (see Figure 4.5).

We now define our instance of Partial Digest With Errors and show at the same time how to construct a solution for this instance. Let

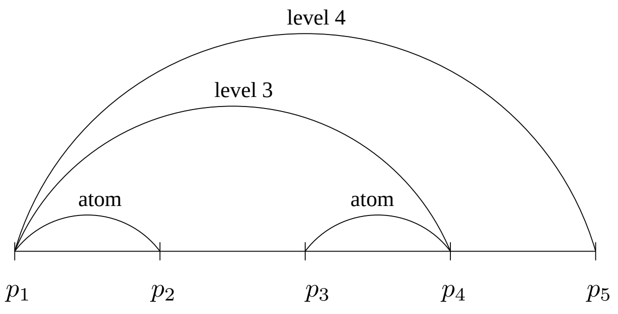


Figure 4.5: Distances of different level.

$c = n^2 \cdot h^2$. Moreover, define error $\varepsilon := 3H$. The distances are expressed as numbers with base $Z = 10nc$, and each distance consists of three digits. The first digit will denote the *level* of a distance (the meaning of the other two digits will become clear soon).

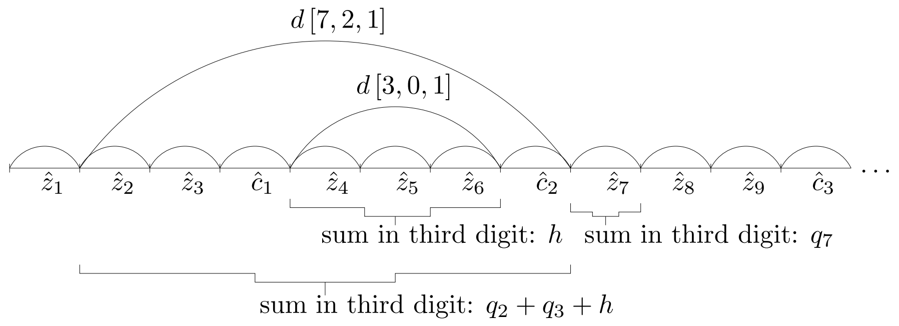
First we define $4n - 1$ distances that will turn out to be atoms in our solution:

$$\begin{aligned} z_i &= \langle 1, 0, q_i \rangle - \varepsilon && \text{for } 1 \leq i \leq 3n, \text{and} \\ c_i &= \langle 1, c, 0 \rangle - \varepsilon && \text{for } 1 \leq i \leq n - 1. \end{aligned}$$

Observe that the operation "$-\varepsilon$" only affects the last digit (and in fact, we could have defined z_i by $\langle 1, 0, q_i - \varepsilon \rangle$ instead), since we choose base Z sufficiently large.

Using these distances, we can already define a "solution" P for distance multiset D (although we did not finish yet to define D; in fact, we will construct D in the following such that it matches point set P up to error ε): Let $\hat{z}_i = z_i + \varepsilon$ for $1 \leq i \leq 3n$, and $\hat{c}_i = c_i + \varepsilon$ for $1 \leq i \leq n - 1$. Observe that each $\hat{z}_i$ has exactly value q_i in its third digit. We call these values *z–pseudoatoms* or *c–pseudoatoms*, respectively, and use them to define a point set $P = \{p_1, \ldots, p_{4n}\}$ by specifying the pairwise distances between the points: Starting in 0, the points have distances $\hat{z}_1$, $\hat{z}_2$, $\hat{z}_3$, $\hat{c}_1$, $\hat{z}_4$, $\hat{z}_5$, $\hat{z}_6$, $\hat{c}_2, \ldots, \hat{c}_{n-1}$, $\hat{z}_{3n-2}$, $\hat{z}_{3n-1}$, $\hat{z}_{3n}$, i.e., we alternate blocks of three z–pseudoatoms and one c–pseudoatom, starting and ending with a block of three z–pseudoatoms (see Figure 4.6).

We now show level by level how the distances in D are defined, and that error ε (which is $3H$) is sufficient to make all distances from D match some distance between points in P.



$d\,[7,2,1]$
$d\,[3,0,1]$
$\hat{z}_1$ $\hat{z}_2$ $\hat{z}_3$ $\hat{c}_1$ $\hat{z}_4$ $\hat{z}_5$ $\hat{z}_6$ $\hat{c}_2$ $\hat{z}_7$ $\hat{z}_8$ $\hat{z}_9$ $\hat{c}_3$ $\dots$
sum in third digit: h
sum in third digit: q_7
sum in third digit: $q_2 + q_3 + h$

Figure 4.6: Atoms and distances in multiset D.

By construction of P, the distances of level 1 are the pseudoatoms, and they match the corresponding z_i's and c_i's up to error ε.

To denote the distances of higher levels we use notation $d\,[\ell, j, k]$, for appropriate parameters ℓ, j and k. These names already indicate the values of the three digits of a distance: Distance $d\,[\ell, j, k]$ will have value ℓ in the first digit, which will be the level of the distance in our point set P. The second digit of the distance has value $j \cdot c$, which denotes that this distance will be used to span j c–pseudoatoms (and $\ell - j$ z–pseudoatoms) in our point set P. For instance, in Figure 4.6 distance $d[7, 2, 1]$ spans the two pseudoatoms $\hat{c}_1$ and $\hat{c}_2$ (and five $\hat{z}_i$'s). Finally, the third digit of distance $d\,[\ell, j, k]$ has value $k \cdot h$ plus some "small offset", which will be a multiple of H. Here, k specifies how many *complete* blocks of three adjacent z–pseudoatoms the distance spans in P (recall that such a block corresponds to three q_i's that sum up to exactly h). In the following, we show how to choose these offsets in the third digit such that our point set P matches distance multiset D up to error ε.

First consider distances of level 2 in P, i.e., two points $p_i, p_{i+2} \in P$ with one point p_{i+1} in between. There are four possibilities for the two pseudoatoms between these two points, for some $0 \leq k \leq n - 1$:

- Case 1: $\hat{z}_{3k+1}$ and $\hat{z}_{3k+2}$;
- Case 2: $\hat{z}_{3k+2}$ and $\hat{z}_{3k+3}$;
- Case 3: $\hat{z}_{3k+3}$ and $\hat{c}_k$; or
- Case 4: $\hat{c}_k$ and $\hat{z}_{3k+1}$.

For the first case, the two pseudoatoms sum up to 2 in the first and to 0 in the second digit. For the third digit of the sum, recall that $\hat{z}_{3k+1}$

has value q_{3k+1} in its third digit, and $\hat{z}_{3k+2}$ has value q_{3k+2} in its third digit. Hence, inequalities (4.2) yield that the third digit of $\hat{z}_{3k+1} + \hat{z}_{3k+2}$ is bounded below by $6H$ and bounded above by $8H$. We define a distance $d\,[2,0,0] := \langle 2,0,9H\rangle$. Obviously, we can span the two pseudoatoms by this distance if we apply at most error ε (recall that $\varepsilon = 3H$). Observe that we could have chosen other values for the third digit of $d\,[2,0,0]$, namely any value between $5H$ and $9H$ (which still allows to match the bounds using error ε). Here, we chose value $9H$, since we will use that same distance to cover the two pseudoatoms in Case 2 as well (see below).

Case 1 occurs exactly n times in our point set P, once for each block of three z–pseudoatoms. Hence, we let distance $d\,[2,0,0]$ be n times in our distance multiset D.

Case 2 is similar to Case 1: The third digit of $\hat{z}_{3k+2} + \hat{z}_{3k+3}$ is bounded below by $8H$ and bounded above by $12H$, using again inequalities (4.2). Like before, this case occurs n times, and we can use n *additional* distances $d\,[2,0,0]$ in D to span such two pseudoatoms up to error ε. Thus, in total we have $2n$ distances $d\,[2,0,0]$ in D that arise from the first two cases.

For the remaining two cases of two pseudoatoms, the last digit of the two pseudoatoms is at least $4H$ and at most $6H$ in Case 3, and at least $3H$ and at most $4H$ in Case 4. Moreover, in both cases the first digit of the sum is 2 and the second digit is c, and both cases occur exactly $n-1$ times. Hence, we can define distance $d\,[2,1,0] := \langle 2,c,4H\rangle$ and include it $2(n-1)$ times in D, in order to cover these pairs of pseudoatoms, again up to error ε.

Before we specify the distances of higher level, we introduce a graphical representation of pseudoatoms: Each z–pseudoatom is represented by a $\bullet$, and each c–pseudoatom by a $|$. This allows us to depict sequences of pseudoatoms without referring to their exact names. E.g. pseudoatoms $\hat{z}_3\hat{c}_1\hat{z}_4\hat{z}_5\hat{z}_6\hat{c}_2$ yield $\bullet|\bullet\bullet\bullet|$, and the four cases of two adjacent pseudoatoms above can be represented by $\bullet\bullet$, $\bullet\bullet$, $\bullet|$ and $|\bullet$.

We now define the distances of higher level. Analogously to distances of level 2, we can compute for each level the corresponding upper and lower bounds for the third digit and define appropriate distances in D. Figure 4.7 shows the distances and multiplicities for level 2 to 7. This table is organized as follows. The first column specifies the level of the distance, and the second column gives the graphical representation of the combinations of pseudoatoms that can occur. The next column specifies how often each combination occurs, and the following two columns show lower and upper bounds for the third digit of the sum of the pseudoatoms. Finally, the last two columns specify the distance name that is used to cover the pseudoatoms, and the value of the distance. Distance values are only intro-

level ℓ	pseudo-atoms	multi-plicity	lower bound	upper bound	distance name	distance value
2	●●	n	$6H$	$8H$	$d\,[2,0,0]$	$\langle 2,0,9H\rangle$
	●●	n	$8H$	$12H$	$d\,[2,0,0]$	
	●|	$n-1$	$4H$	$6H$	$d\,[2,1,0]$	$\langle 2,c,4H\rangle$
	|●	$n-1$	$3H$	$4H$	$d\,[2,1,0]$	
3	●●●	n	$12H$	$12H$	$d\,[3,0,1]$	$\langle 3,0,12H\rangle+\varepsilon$
	|●●	$n-1$	$6H$	$8H$	$d\,[3,1,0]$	$\langle 3,c,9H\rangle$
	●|●	$n-1$	$7H$	$10H$	$d\,[3,1,0]$	
	●●|	$n-1$	$8H$	$12H$	$d\,[3,1,0]$	
4	●●|●	$n-1$	$11H$	$16H$	$d\,[4,1,0]$	$\langle 4,c,13H\rangle$
	●|●●	$n-1$	$10H$	$14H$	$d\,[4,1,0]$	
	●●●|	$n-1$	$12H$	$12H$	$d\,[4,1,1]$	$\langle 4,c,12H\rangle$
	|●●●	$n-1$	$12H$	$12H$	$d\,[4,1,1]$	
5	●●|●●	$n-1$	$14H$	$20H$	$d\,[5,1,0]$	$\langle 5,c,17H\rangle$
	●●●|●	$n-1$	$15H$	$16H$	$d\,[5,1,1]$	$\langle 5,c,16H\rangle$
	●|●●●	$n-1$	$16H$	$18H$	$d\,[5,1,1]$	
	|●●●|	$n-2$	$12H$	$12H$	$d\,[5,2,1]$	$\langle 5,2c,12H\rangle$
6	●●●|●●	$n-1$	$18H$	$20H$	$d\,[6,1,1]$	$\langle 6,c,21H\rangle$
	●●|●●●	$n-1$	$20H$	$24H$	$d\,[6,1,1]$	
	●|●●●|	$n-2$	$16H$	$18H$	$d\,[6,2,1]$	$\langle 6,2c,16H\rangle$
	|●●●|●	$n-2$	$15H$	$16H$	$d\,[6,2,1]$	
7	●●●|●●●	$n-1$	$24H$	$24H$	$d\,[7,1,2]$	$\langle 7,c,24H\rangle$
	●●|●●●|	$n-2$	$20H$	$24H$	$d\,[7,2,1]$	$\langle 7,2c,21H\rangle$
	●|●●●|●	$n-2$	$19H$	$22H$	$d\,[7,2,1]$	
	|●●●|●●	$n-2$	$18H$	$20H$	$d\,[7,2,1]$	

Figure 4.7: Distances up to level 7.

duced once, and the lines are sorted such that those cases that use the same distance stand together.

Observe that $d\,[2,0,0]$ and $d\,[6,1,1]$ are in a sense “equivalent”, since they are used for cases that differ only in one complete block of three z–pseudoatoms and one c–pseudoatom. Hence, we could replace the definition by $d\,[6,1,1]=d\,[2,0,0]+\langle 4,c,h\rangle$. Moreover, $d\,[6,2,1]=d\,[2,1,0]+\langle 4,c,h\rangle$

level ℓ	pseudo-atoms	multi-plicity	distance name	distance value
$4k+4$	$\bullet\bullet\mid\dots\mid\bullet$	$n-k-1$	$d\,[4+4k, 1+k, 0+k]$	$d\,[4,1,0]+k\cdot\beta$
	$\bullet\mid\dots\mid\bullet\bullet$	$n-k-1$	$d\,[4+4k, 1+k, 0+k]$	
	$\bullet\bullet\bullet\mid\dots\mid$	$n-k-1$	$d\,[4+4k, 1+k, 1+k]$	$d\,[4,1,1]+k\cdot\beta$
	$\mid\dots\mid\bullet\bullet\bullet$	$n-k-1$	$d\,[4+4k, 1+k, 1+k]$	
$5+4k$	$\bullet\bullet\mid\dots\mid\bullet\bullet$	$n-k-1$	$d\,[5+4k, 1+k, 0+k]$	$d\,[5,1,0]+k\cdot\beta$
	$\bullet\bullet\bullet\mid\dots\mid\bullet$	$n-k-1$	$d\,[5+4k, 1+k, 1+k]$	$d\,[5,1,1]+k\cdot\beta$
	$\bullet\mid\dots\mid\bullet\bullet\bullet$	$n-k-1$	$d\,[5+4k, 1+k, 1+k]$	
	$\mid\dots\mid\bullet\bullet\bullet\mid$	$n-k-2$	$d\,[5+4k, 2+k, 1+k]$	$d\,[5,2,1]+k\cdot\beta$
$6+4k$	$\bullet\bullet\bullet\mid\dots\mid\bullet\bullet$	$n-k-1$	$d\,[6+4k, 1+k, 1+k]$	$d\,[6,1,1]+k\cdot\beta$
	$\bullet\bullet\mid\dots\mid\bullet\bullet\bullet$	$n-k-1$	$d\,[6+4k, 1+k, 1+k]$	
	$\bullet\mid\dots\mid\bullet\bullet\bullet\mid$	$n-k-2$	$d\,[6+4k, 2+k, 1+k]$	$d\,[6,2,1]+k\cdot\beta$
	$\mid\dots\mid\bullet\bullet\bullet\mid\bullet$	$n-k-2$	$d\,[6+4k, 2+k, 1+k]$	
$7+4k$	$\bullet\bullet\bullet\mid\dots\mid\bullet\bullet\bullet$	$n-k-1$	$d\,[7+4k, 1+k, 2+k]$	$d\,[7,1,2]+k\cdot\beta$
	$\bullet\bullet\mid\dots\mid\bullet\bullet\bullet\mid$	$n-k-2$	$d\,[7+4k, 2+k, 1+k]$	$d\,[7,2,1]+k\cdot\beta$
	$\bullet\mid\dots\mid\bullet\bullet\bullet\mid\bullet$	$n-k-2$	$d\,[7+4k, 2+k, 1+k]$	
	$\mid\dots\mid\bullet\bullet\bullet\mid\bullet\bullet$	$n-k-2$	$d\,[7+4k, 2+k, 1+k]$	

Figure 4.8: Distances with level 8 to $4n-5$. Value k varies between 1 and $n-3$.

and $d\,[7,2,1] = d\,[3,1,0] + \langle 4, c, h\rangle$. Similarly, distances of level greater than 7 can be decomposed into a distance of low level (4 to 7) and an appropriate number of blocks of three z–pseudoatoms and one c–pseudoatom. We set $\beta := \langle 4, c, h\rangle$ and define in Figure 4.8 the distances of level 8 to $4n-5$. In the table, the number of blocks k varies from 1 to $n-3$. Finally, in Figure 4.9 the distances that have level $4n-4$ to $4n-1$ are shown. Observe that as before they are derived from distances of level 4 to 7, for $k = n-2$. However, not all combinations are necessary for these distances.

Our distance multiset D consists of all atoms z_i and c_i, and all distances specified in Figures 4.7, 4.8 and 4.9, with the corresponding multiplicities. There are $4n-1$ levels, and for each level ℓ there are $4n-\ell$ distances in D. In total, this yields $\sum_{\ell=1}^{4n-1}(4n-\ell) = \binom{4n}{2}$ distances. The cardinality of D is polynomially bounded in n, and each distance in D is polynomial in h. Hence, multiset D can be constructed in polynomial time from a given instance of 3–Partition.

level ℓ	lower bound	upper bound	distance name	distance value
$4n-4$	$(n-2)h+11H$	$(n-2)h+16H$	$d\,[4n-4, n-1, n-2]$	$d\,[4,1,0]+(n-2)\cdot\beta$
	$(n-2)h+10H$	$(n-2)h+14H$	$d\,[4n-4, n-1, n-2]$	
	$(n-1)h$	$(n-1)h$	$d\,[4n-4, n-1, n-1]$	$d\,[4,1,1]+(n-2)\cdot\beta$
	$(n-1)h$	$(n-1)h$	$d\,[4n-4, n-1, n-1]$	
$4n-3$	$(n-1)h+3H$	$(n-1)h+4H$	$d\,[4n-3, n-1, n-1]$	$d\,[5,1,1]+(n-2)\cdot\beta$
	$(n-1)h+4H$	$(n-1)h+6H$	$d\,[4n-3, n-1, n-1]$	
	$(n-2)h+14H$	$(n-2)h+20H$	$d\,[4n-3, n-1, n-2]$	$d\,[5,1,0]+(n-2)\cdot\beta$
$4n-2$	$(n-1)h+6H$	$(n-1)h+8H$	$d\,[4n-2, n-1, n-1]$	$d\,[6,1,1]+(n-2)\cdot\beta$
	$(n-1)h+8H$	$(n-1)h+12H$	$d\,[4n-2, n-1, n-1]$	
$4n-1$	nh	nh	$d\,[4n-1, n-1, n]$	$\langle 4n-1, (n-1)c, nh\rangle+\varepsilon$

Figure 4.9: Distances with level $4n-4$ to $4n-1$. Each case occurs once.

Observe that the construction of D is possible for *any* instance of 3–Partition, and does *not* rely on the fact that there is a solution for the 3–Partition instance, nor on a particular ordering of the q_i's. In our argumentation above, we used these two properties of the instance only to construct simultaneously a point set P that matches D up to error ε. Hence, we have constructed an instance D and ε of Partial Digest With Errors from the given instance of 3–Partition, and we have shown already that a solution for the 3–Partition instance yields a solution for the Partial Digest With Errors instance.

In the following, we show the opposite direction, i.e., we show that a solution for the Partial Digest With Errors instance yields a solution for the 3–Partition instance.

Let $R=\{r_1,\ldots,r_{4n}\}$ be *any* set of $4n$ points on a line that is a solution for the Partial Digest With Errors instance, i.e., multiset D is the multiset of pairwise distances of R, up to error ε for each distance. We assume w.l.o.g. that the points are ordered from left to right, i.e., $r_1 < r_2 < \ldots < r_{4n}$. We will show that R is basically identical to P, the point set that we constructed above.

Obviously, error ε can affect only the last digit of each distance, since base Z is sufficiently large. Thus, exactly those distances with value 1 in the first digit are atoms, since all other distances have value greater than 1 in the first digit, and since there must be exactly $4n-1$ atoms. This implies immediately that the first digit of each distance denotes the level of the distance in any solution.

We now show that error $+\varepsilon$ has to be applied to each single atom to make it fit to the distances between adjacent points in R. To see this, first observe that the atoms sum up to

$$\begin{aligned}
& \sum_{i=1}^{3n} z_i + \sum_{i=1}^{n-1} c_i \\
= \; & \sum_{i=1}^{3n} (\langle 1, 0, q_i \rangle - \varepsilon) + \sum_{i=1}^{n-1} (\langle 1, c, 0 \rangle - \varepsilon) \\
= \; & \langle 3n, 0, nh \rangle - 3n\varepsilon + \langle n-1, (n-1)c, 0 \rangle - (n-1)\varepsilon \\
= \; & \langle 4n-1, (n-1)c, nh \rangle - (4n-1)\varepsilon .
\end{aligned}$$

On the other hand, the largest distance in multiset D is $d\,[4n-1, n-1, n] = \langle 4n-1, (n-1)c, nh \rangle + \varepsilon$. Each atom is the distance between two adjacent points in R, up to error ε, while $d\,[4n-1, n-1, n]$ is the distance between the first and the last point in R, again up to error ε. Hence, the atoms must sum up to the length of the largest distance. This is only possible if we apply error $+\varepsilon$ to each atom, yielding sum $\langle 4n-1, (n-1)c, nh \rangle$, and if we apply error $-\varepsilon$ to the largest distance, yielding $\langle 4n-1, (n-1)c, nh \rangle$ as well. Knowing this, we can again define *pseudoatoms* $\hat{z}_i = z_i + \varepsilon$ and $\hat{c}_i = c_i + \varepsilon$, which represent exactly the distances of adjacent points in R (without error). Observe that if we represented the distances between adjacent points in R in our number representation, then pseudoatom $\hat{z}_i$ would have exactly value q_i in its last digit, for all $1 \le i \le 3n$.

We now show that the ordering of the pseudoatoms arising from R is such that there are n blocks of three pseudoatoms $\hat{z}_i$, and each two blocks are separated by one pseudoatom $\hat{c}_i$. Again, we call the pseudoatoms with value c in the second digit c–pseudoatoms, and those with value 0 in the second digit are called z–pseudoatoms. Between any two adjacent c–pseudoatoms there must be exactly three z–pseudoatoms: Since there are no distances of level 4 with value $2c$ in the second digit, no combination $||$ or $| \bullet |$ or $| \bullet \bullet |$ is possible, and there are at least three z–pseudoatoms in between two c–pseudoatoms; moreover, since there are $n-2$ distances of level 5 with value $2c$ in the second digit, there must be at least $n-1$ c–pseudoatoms such that there are always at most 3 z–pseudoatoms in between. Hence, the points in R are such that blocks of three z–pseudoatoms alternate with one c–pseudoatom, starting and ending with a block of three z–pseudoatoms.

Finally, we show that the third digits of each three adjacent z–pseudoatoms sum up to h: Consider those distances of level 3 that have a zero in the second digit. There are n such distances, and their third digits sum up to $nh + n\varepsilon$. Each of these distances must span exactly one of the n blocks of three z–pseudoatoms. The total sum of the last digit of all z–pseudoatoms is exactly $\sum_{i=1}^{3n} q_i = nh$. Since the distances of level 3 that span these blocks do not overlap, they have to sum up to the same total. Hence, the error for each such distance of level 3 must be $-\varepsilon$. This implies that each three q_i's that correspond to one block sum up to exactly h (since we have applied error $+\varepsilon$ to each atom to define the z–pseudoatoms). Thus, these triples yield a solution for the 3–Partition instance.

□

4.5 Conclusion

We have shown that the minimization problem Min Partial Digest Superset is NP-hard, and that the maximization problem Max Partial Digest Subset is hard to approximate. This answers open problem 12.116 left open in the book by Pevzner [71]. Moreover, we have shown that Partial Digest is strongly NP-complete if all measurements are prone to the same additive error. However, in the realm of Partial Digest, many questions are still open:

- Since our optimization variations model different error types that (always) occur in real–life data, our hardness results suggest that real–life Partial Digest problems are in fact instances of NP-hard problems. However, the backtracking algorithm from [55] performs well in experiments [100]. How can this be explained?
- What is the best approximation ratio for Min Partial Digest Superset?
- In our NP-hardness proof for Partial Digest With Errors, we used non–constant error $\varepsilon = \frac{h}{4}$. Is Partial Digest still NP-complete if we restrict the error to some (small) constant? What if we allow only one–sided errors, i.e., if the lengths of the distances are for instance always underestimated? And what is the complexity of Partial Digest if we have a (fixed) relative error, i.e., if the error is some fixed percentage of the distance length?
- Using gel electrophoresis, it is very hard to determine the correct multiplicity of a distance. This yields the following variation of Partial Digest: We are given a *set* of distances, and for each distance a multiplicity, and we ask for points on a line such that the multiplicities of the corresponding distance set do not differ "too much" from the given multiplicities. What is the computational complexity of this problem?

- Is there a polynomial–time algorithm for the Partial Digest problem if we restrict the input to be a *set* of distances (instead of a multiset), i.e., if we know in advance that each two distances in the input are pairwise distinct?

Finally and obviously, the main open problem is of course the computational complexity of Partial Digest itself.

Chapter 5

Equal Sum Subsets

5.1 Introduction

In this chapter, we study the complexity of variations of Equal Sum Subsets. We recapitulate the definition (cf. Definition 1.4.2):

Definition. *Given a set A of n positive integers, are there two disjoint non–empty subsets $X, Y \subseteq A$ such that* $\text{sum}(X) = \text{sum}(Y)$*?*

In the previous chapter, we used a reduction from Equal Sum Subsets to show NP-hardness of Min Partial Digest Superset (cf. Theorem 4.2.1). The problem Equal Sum Subsets is a relaxation of Partition in the sense that we do not require the two subsets to cover all input numbers. The problem can be also seen as a variation of Bin Packing with fixed number of bins, where we require that all bins should be filled to the same level, while it is not necessary to use all the elements. While Equal Sum Subsets, Partition, Bin Packing and their variations have numerous applications in production planning and scheduling (see for instance the book by Martello and Toth for a survey [60]), our interest in Equal Sum Subsets comes from its relation to Partial Digest, since studying the computational complexity of Equal Sum Subsets and its variations might yield new insight into the complexity of Partial Digest as well. For this reason, we study the complexity of Equal Sum Subsets variations here, although they are only loosely connected to bioinformatics.

Only little is known about Equal Sum Subsets: It is NP-complete [93], and there exists an FPTAS for the optimization version of Equal Sum Subsets in which the ratio of the sums of the two subsets is to be minimized [7]. Obviously, if the sum of the n given numbers is at most

$2^n - 1$, then at least two of the 2^n possible subsets of the numbers must have equal sum, hence the decision version of Equal Sum Subsets becomes trivial. In this case, the problem has been studied in the context of function problems [65]. In order to better understand Equal Sum Subsets as a combinatorial problem, we extensively study different variations of Equal Sum Subsets.

In the first set of Equal Sum Subsets variations that we study, we ask for two subsets such that the *ratio* of their sums is exactly r, for some fixed rational $r > 0$. We call this problem Factor–r Sum Subsets and define it as follows.

Definition 5.1.1 (Factor–r Sum Subsets)**.** *Given a set A of n positive integers, are there two disjoint non–empty subsets $X, Y \subseteq A$ such that* $\text{sum}(X) = r \cdot \text{sum}(Y)$*?*

This problem is very closely related to the minimization version of Equal Sum Subsets studied in [7]. In Section 5.2, we show that Factor–r Sum Subsets is NP-complete for any rational factor $r > 0$ by giving two reductions from Exact 3–Satisfiability, one that works for all $r > 0$ with $r \notin \{1, 2, \frac{1}{2}\}$, and one that works for the cases $r = 2$ and $r = \frac{1}{2}$. The case $r = 1$ is just Equal Sum Subsets.

The second generalization of Equal Sum Subsets that we study is the problem k Equal Sum Subsets, in which we need to find k (disjoint) subsets of equal sum from a given set of numbers, for given $k \geq 2$:

Definition 5.1.2 (k Equal Sum Subsets)**.** *Given a multiset of n positive integers $A = \{a_1, \ldots, a_n\}$, are there k disjoint non–empty subsets $S_1, \ldots, S_k \subseteq \{a_1, \ldots, a_n\}$ such that* $\text{sum}(S_1) = \ldots = \text{sum}(S_k)$*?*

Observe that we allow multisets here, in contrast to Equal Sum Subsets, which becomes trivial if any number occurs more than once. Obviously, if $k = 2$ and the input is a set instead of a multiset, then k Equal Sum Subsets is just Equal Sum Subsets. If we require that our subsets yield a full partition of the given numbers, our problem would turn into a variation of Partition with k sets instead of 2.

We first show in Section 5.3.1 that k Equal Sum Subsets is NP-complete for any integer $k \geq 3$ by giving a reduction from Alternating Partition, which is an NP-complete variation of Partition [40].

Then we study the influence of parameter k on the complexity of k Equal Sum Subsets in more depth. We have introduced parameter k for the number of equal size subsets as a fixed constant that is part of the problem definition. An interesting variation is to allow k to be a (fixed) function of the number of input elements n, e.g. $k = \frac{n}{q}$ for some constant q. In the sequel, we will always consider k as a function of n; whenever k is a constant we simply write $k = O(1)$. In Section 5.3.2, we present a dynamic programming algorithm for k Equal Sum Subsets with running time $O(\frac{nS^k}{k^{k-1}})$, where n is the cardinality of the input set and S is the sum of all numbers in the input set; the algorithm runs in pseudo–polynomial time for $k = O(1)$. On the other hand, we show that k Equal Sum Subsets is strongly NP-complete for $k = \Omega(n)$. We obtain this result by giving a reduction from 3–Partition.

The definition of k Equal Sum Subsets corresponds to the situation in which it is allowed to form subsets that do not have the same number of elements. In some cases, this makes sense; however, we may also wish to have the same number of elements in each subset. Such problems occur for instance when we are given a set of, say, soccer players, together with their strength, and we want to compose teams of equal strength and size to play a tournament. In Section 5.3.3, we study three variations of k Equal Sum Subsets with *equal* cardinalities, where either we specify the cardinality of the subsets in the input, or the cardinality is a fixed constant (part of the problem definition), or we only ask for subsets of equal cardinality, but do not specify their cardinality at all. The corresponding problems are defined as follows.

Definition 5.1.3 (kESS Of Cardinality c)**.** *Given a multiset A of n positive integers, are there k disjoint non–empty subsets $S_1, \ldots, S_k \subseteq A$ with sum$(S_1) = \ldots =$ sum(S_k) such that each S_i has cardinality c?*

Definition 5.1.4 (kESS Of Specified Cardinality)**.** *Given a multiset A of n positive integers and a cardinality c, are there k disjoint non–empty subsets $S_1, \ldots, S_k \subseteq A$ with sum$(S_1) = \ldots =$ sum(S_k) such that each S_i has cardinality c?*

Definition 5.1.5 (kESS Of Equal Cardinality)**.** *Given a multiset A of n positive integers, are there k disjoint non–empty subsets $S_1, \ldots, S_k \subseteq A$ with sum$(S_1) = \ldots =$ sum(S_k) such that all S_i's have the same cardinality?*

In Section 5.3.3, we first present a polynomial time algorithm for kESS Of Cardinality c. The algorithm uses exhaustive search and runs in time $O(n^{kc})$, which is polynomial in n as the two parameters k and c are fixed constants. On the other hand, we show that kESS Of Specified Cardinality is NP-complete. To establish this result, we present a reduction from Alternating Partition. A similar reduction can be used to prove NP-completeness for kESS Of Equal Cardinality. However, we show that none of these two problems is strongly NP-complete by presenting an algorithm that can solve them in pseudo–polynomial time.

After that we come back to the case where we ask for only two equal sum subsets. In many settings, it is required that the two equal sum subsets fulfill additional requirements. One such requirement is that the subsets have to respect a given set of exclusions, for instance if we want to find groups of people for medical experiments that fulfill some restrictions. This yields the following problem.

Definition 5.1.6 (ESS With Exclusions)**.** *Given a set A of n positive integers and an* exclusion graph $G_{ex} = (A, E_{ex})$ *with vertices A and edges $E_{ex} \subseteq A \times A$. Are there two disjoint non–empty subsets $X, Y \subseteq A$ with* $\text{sum}(X) = \text{sum}(Y)$ *such that each of the two sets is an independent set in G_{ex}, i.e., there is no edge between any two vertices in X or any two vertices in Y?*

This problem is in a sense a generalization of the Party Invitation problem, where we are given a tree (the hierarchical structure of a company) and for each node a value (a conviviality rating of an employee), and we want to find a set of nodes (people to be invited) of maximum sum such that there is no node and its parent node in the set (no employee and its supervisor). This is a standard example for dynamic programming and can be solved in polynomial time [27].

The problem ESS With Exclusions is obviously NP-complete, since it is just Equal Sum Subsets if the exclusion graph is empty. We give a pseudo–polynomial time algorithm for this problem in Section 5.4.1. If we want to model preferences, we can

If we do not want to exclude elements, but on the contrary we want to ensure that some numbers of the input *occur* in the subsets, then this yields the following two problems: In ESS With Enforced Element we enforce one element, say the last, of the input numbers to be in one of the subsets; in Alternating Equal Sum Subsets we have for each input number a “partner”, and if a number occurs in one set, then its partner has to be in the other set. This problem is the “partial” equivalent of Alternating Partition. More formally, the two problems are defined as follows.

Definition 5.1.7 (ESS With Enforced Element)**.** *Given a set $A = \{a_1, \dots, a_n\}$ of n positive integers, are there two disjoint subsets $X, Y \subseteq A$ with* $\text{sum}(X) = \text{sum}(Y)$ *such that $a_n \in X$?*

Definition 5.1.8 (Alternating Equal Sum Subsets)**.** *Given n pairs of positive integers $(u_1, v_1), \dots, (u_n, v_n)$, are there two disjoint nonempty sets of indices I and J such that $\sum_{i \in I} u_i + \sum_{j \in J} v_j = \sum_{i \in I} v_i + \sum_{j \in J} u_j$?*

We show in Section 5.4.1 that both problems above are NP-complete, by reducing Alternating Partition to the former and Equal Sum Subsets to latter, respectively.

We then study variations of Equal Sum Subsets where we restrict the cardinality of the equal sum subsets. If we ask for equal cardinalities, then the corresponding results for k Equal Sum Subsets apply. On the other hand, if we want the cardinalities to be different, this yields the following problem definition.

Definition 5.1.9 (ESS Of Different Cardinality)**.** *Given a set A of n positive integers, are there two disjoint subsets $X, Y \subseteq A$ with* $\mathrm{sum}(X) = \mathrm{sum}(Y)$ *such that $|X| \neq |Y|$?*

We show that ESS Of Different Cardinality is NP-complete, and that it remains hard to solve even if we specify the difference of the two set cardinalities.

Finally, we turn to the variation Equal Sum Subsets where we are given *two* sets of numbers and we ask for two equal sum subsets, one from each set. The corresponding problem ESS From Two Sets is defined as follows.

Definition 5.1.10 (ESS From Two Sets)**.** *Given two sets A and B of positive integers, are there two nonempty subsets $U \subseteq A$ and $V \subseteq B$ such that* $\mathrm{sum}(U) = \mathrm{sum}(V)$*?*

We first show in Section 5.4.3 that the problem ESS From Two Sets is NP-complete. Then we study the following four variations of the problem where we restrict the choice of elements in the subsets:

Definition 5.1.11. *Given two sets of positive integers $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$, are there two nonempty indices sets $I, J \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{j \in J} b_j$, and that comply with the following additional condition:*

ESS Of Equal Cardinality From Two Sets*: $|I| = |J|$*
ESS With Disjoint Indices From Two Sets*: $I \cap J = \emptyset$*
ESS With Disjoint Covering Indices From Two Sets*: $I \cap J = \emptyset$ and $I \cup J = \{1, \ldots, n\}$*
ESS With Identical Indices From Two Sets*: $I = J$*

We show in Section 5.4.3 that each of these problem variations is NP-complete.

Part of the results in this chapter have been published previously [18, 19].

5.2 NP-Completeness of Factor–r Sum Subsets

In this section, we study the Factor–r Sum Subsets problem. For $r = 1$, the problem is Equal Sum Subsets and therefore NP-complete [93]. We show that Factor–r Sum Subsets is actually NP-complete for any fixed rational $r > 0$. The proof of NP-hardness consists of two different reductions from Exact 3–Satisfiability (see Definition 2.4.4), where the second reduction is just for the cases $r = 2$ and $r = \frac{1}{2}$.

Lemma 5.2.1. *Factor–r Sum Subsets is NP-hard for any rational $r > 0$ with $r \notin \{1, 2, \frac{1}{2}\}$.*

Proof: We present a reduction from Exact 3–Satisfiability. Let $r = p/q$, where p and q are positive integers with no common divisor except 1 (coprimes) and $p < q$. [The case $p > q$ is equivalent by interchanging sets X and Y in the problem definition.] We distinguish several cases, depending on the values of p and q. We only give a detailed proof for the first case; for the other cases the proof is quite similar, so we just mention the construction of the necessary numbers.

Case 1: $p > 3$. Consider an instance of Exact 3–Satisfiability with a set of n variables $V = \{v_1, \ldots, v_n\}$ and a set of m clauses $C = \{c_1, \ldots, c_m\}$. An instance of Factor–r Sum Subsets is constructed as follows. For each variable v_i a number $a_i = \sum_{v_i \in c_j} \Delta_m(j)$ is defined. Value a_i has m digits, and its non–zero digits correspond to clauses where v_i appears. Two additional numbers a_{n+1} and a_{n+2} are constructed which are multiples of $\mathbb{1}_m$: $a_{n+1} = (p-1) \cdot \mathbb{1}_m$ and $a_{n+2} = q \cdot \mathbb{1}_m$. For all numbers we use base $Z = q(p+q+2)+1$. This way we will avoid carry–overs from one digit to the next when adding a_i's. Let $A = \{a_1, \ldots, a_{n+2}\}$. In the following, we show that there is a solution for the Exact 3–Satisfiability instance if and only if there are two disjoint nonempty subsets $X, Y \subseteq A$ such that $\text{sum}(X) = r \cdot \text{sum}(Y)$.

"only if": Assume that there exists an exact satisfying assignment for the clauses in C. This implies that there exists a subset $R \subseteq \{a_1, \ldots, a_n\}$ such that $\text{sum}(R) = \mathbb{1}_m$, since for each clause c_j there is exactly one of the three variables in c_j set to TRUE, say v_k, and the corresponding a_k has a 1 in the j–th digit. We define a set R to contain exactly these a_i's; then $\text{sum}(R) = \mathbb{1}_m$. Hence, by setting $X = R \cup \{a_{n+1}\}$ and $Y = \{a_{n+2}\}$, we have $\text{sum}(X) = p \cdot \mathbb{1}_m = r \cdot q \cdot \mathbb{1}_m = r \cdot \text{sum}(Y)$, thus X and Y yield a solution for the Factor–r Sum Subsets instance.

"if": For the opposite direction, assume that non–empty sets X, Y exist such that $\text{sum}(X) = r \cdot \text{sum}(Y)$; equivalently, $q \cdot \text{sum}(X) = p \cdot \text{sum}(Y)$.

Observe that summing the i'th digit of all numbers in the input set A yields $p+q+2$. Moreover, even when multiplying each number in A by q we get only total $q(p+q+2)$ in the i'th digit, and no carry–overs occur, since we choose base Z sufficiently large. Since $q \cdot \mathrm{sum}\,(X) = p \cdot \mathrm{sum}\,(Y)$, we have $qx_i = py_i$, where x_i and y_i are the i'th digit of $\mathrm{sum}\,(X)$ resp. $\mathrm{sum}\,(Y)$, for $1 \le i \le n$. This implies that for each digit i either $x_i = y_i = 0$, or q divides y_i and p divides x_i (since p and q are coprime). Observe that not all digits can be 0, since we have assumed that X and Y are non–empty.

We now show that $x_j = p$ and $y_j = q$ for every non–zero digit j: Since p divides x_j and q divides y_j, there exist two positive integers k and ℓ such that $x_j = k \cdot p$ and $y_j = \ell \cdot q$. Then $qx_j = py_j$ implies that $k = \ell$. Moreover, we have $p+q+2 \ge x_j + y_j = k(p+q)$, hence $2 \ge (k-1)(p+q)$, and this inequality can only hold for k = 1, since $q > p > 3$ and k is positive. Thus, $x_j = p$ and $y_j = q$.

Since only five numbers in A have non–zero value in the j'th digit, and the corresponding values are $1, 1, 1, p-1$ and q,we can only achieve $x_j = p$ if $X = \{a_{n+1}\} \cup R$, where R is a subset of A such that $\mathrm{sum}\,(R)$ has a 1 in the j'th digit. Thus, the only way to get $y_j = q$ is to have $Y = \{a_{n+2}\}$. Since a_{n+1} has value $p-1$ in every digit, no digits in $\mathrm{sum}\,(X)$ can be 0, hence also in $\mathrm{sum}\,(Y)$. Thus, the variables corresponding to numbers in R form an exact satisfying assignment for the given clauses.

We now sketch the proof for the remaining combinations of values of p and q:

Case 2: $p = 3, q > 4$. Numbers $a_1, \ldots, a_n$ are constructed as in Case 1, $a_{n+1} = 3 \cdot \mathbb{1}_m$, and $a_{n+2} = (q-1) \cdot \mathbb{1}_m$.

Case 3: $p = 3, q = 4$. Numbers $a_1, \ldots, a_n$ are constructed as in Case 1, $a_{n+1} = 3 \cdot \mathbb{1}_m$, and $a_{n+2} = 2 \cdot \mathbb{1}_m$.

Case 4: $p = 2, q > 3$. Numbers $a_1, \ldots, a_n$ are constructed as in Case 1, and only one additional number $a_{n+1} = (q-1) \cdot \mathbb{1}_m$ is used.

Case 5: $p = 2, q = 3$. For each variable v_i let $a_i = \sum_{v_i \in c_j} 3 \cdot \Delta_m(j)$, i.e., a_i has a digit 3 in each position that corresponds to a clause that contains v_i. We also set $a_{n+1} = \mathbb{1}_m$. Note that $\mathrm{sum}\,(A) = 10 \cdot \mathbb{1}_m$. Like in Case 1, the direction "only if" is easy: any exact satisfying assignment for the clauses in C corresponds to numbers a_i that add up to $3 \cdot \mathbb{1}_m$, which together with a_{n+1} constitute X. For the "if" direction, we observe that the only way to have the required ratio is by having two sets X and Y such that $\mathrm{sum}\,(X) = 4 \cdot \mathbb{1}_m$ and $\mathrm{sum}\,(Y) = 6 \cdot \mathbb{1}_m$; this implies $a_{n+1} \in X$, and for each $j \in \{1, \ldots, m\}$ there is exactly one further number $a_i \in X$ that has non–zero digit j. Hence, the variables corresponding to $X - \{a_{n+1}\}$ constitute an exact satisfying assignment.

Case 6: $p = 1, q > 2$. Numbers $a_1, \ldots, a_n$ are constructed as in Case 1, and there is only one additional number $a_{n+1} = q \cdot \mathbb{1}_m$.

□

Lemma 5.2.2. Factor–r Sum Subsets *is* NP*-hard for* $r = 2$.

Proof: We use a restricted, but still NP-hard version of Exact 3–Satisfiability for a reduction to Factor–r Sum Subsets for the case $r = 2$. In the following, let always $r = 2$. Given an Exact 3–Satisfiability instance with variables $v_1, \ldots, v_n$ and clauses $c_1, \ldots, c_m$ with only positive literals, let $G = (V, E)$ be the graph with vertices $V = \{v_1, \ldots, v_n\}$ (i.e., each variable corresponds to a vertex) and, for $i, j \in \{1, \ldots, n\}$, edges $(v_i, v_j) \in E$ if and only if v_i and v_j both occur in a clause c_k, for some $k \in \{1, \ldots, m\}$. The Exact 3–Satisfiability variation in which the corresponding graph G is connected is still NP-hard, because we could use a polynomial algorithm for this variation to solve the unrestricted Exact 3–Satisfiability problem by applying the algorithm to each component of the corresponding graph.

We reduce the restricted variant of Exact 3–Satisfiability with connected graphs to Factor–r Sum Subsets as follows. We construct an instance A of Factor–r Sum Subsets by defining one number a_i for each variable v_i by $a_i := \sum_{v_i \in c_j} \Delta_n(j)$, where we set the j–th digit to 1 if v_i appears as a literal in clause c_j. We let the base Z of these numbers be 7. Observe that among all a_i's there are exactly three ones in each digit.

Assume that we are given an exact satisfying assignment for the variables of the Exact 3–Satisfiability instance. We then construct sets $X, Y \subseteq A$, where Y contains all numbers a_i for which the corresponding variable v_i has been set to TRUE, and X contains all remaining numbers. Thus, $\mathrm{sum}(Y) = \langle 1, 1, \ldots, 1\rangle$ and $\mathrm{sum}(X) = \langle 2, 2, \ldots, 2\rangle$, and therefore, X and Y yield a solution for the Factor–r Sum Subsets instance.

For the opposite direction, assume that we are given a solution X and Y for the Factor–r Sum Subsets instance with $\mathrm{sum}(X) = 2 \cdot \mathrm{sum}(Y)$. Since each digit is set to 1 in exactly three of the numbers a_i, and since no carry–overs can occur when summing up the a_i's because base Z is sufficiently large, $\mathrm{sum}(Y)$ must contain only ones and zeros in its digits, and $\mathrm{sum}(X)$ contains only twos and zeros. Since the sets are not empty, at least one digit must be set to 1. We assign the value TRUE to a variable v_i with corresponding number a_i if $a_i \in Y$, and we assign the value FALSE, if $a_i \in X$. Thus, if a clause $c_j = (v_f, v_g, v_h)$ exists, then either one of the three numbers a_f, a_g, or a_h is in Y and the other two numbers are in X, or neither X nor Y contain a_f, a_g, or a_h. In the latter case, we know that $\mathrm{sum}(X)$ and $\mathrm{sum}(Y)$ would contain a 0 at position j.

However, the numbers $\operatorname{sum}(X)$ and $\operatorname{sum}(Y)$ cannot contain any zero digits because of the connectedness of graph G. In order to see this, assume for the sake of contradiction that $\operatorname{sum}(Y)$ contains some digits that are 0. Then $\operatorname{sum}(X)$ must have digits with value 0 at the same positions. Consider the set S of all variables that occur in clauses which correspond to zero digits in $\operatorname{sum}(X)$ and $\operatorname{sum}(Y)$. Then the subgraph of G with only the vertices corresponding to variables from set S must be a component in the graph G without any edges to other vertices: If such an edge existed, it would imply that the corresponding digit is not set to 0 in either $\operatorname{sum}(X)$ or $\operatorname{sum}(Y)$. To see this, consider an edge $e = (v_f, v_g)$ arising from clause $c_j = (v_f, v_g, v_h)$ with $v_f \in S$ and $v_g \notin S$. Then $a_g \in X \cup Y$, but a_f (and a_h) must be in $X \cup Y$ as well, in order to achieve the factor 2 in the j–th digit.

Thus, there can be no zeroes in any digit in $\operatorname{sum}(X)$ or $\operatorname{sum}(Y)$, and our assignment is a solution for the Exact 3–Satisfiability instance.

□

Since Factor–r Sum Subsets is obviously in NP, Lemmas 5.2.1 and 5.2.2 and the NP-completeness of Equal Sum Subsets yield the following.

Theorem 5.2.3. Factor–r Sum Subsets *is* NP*-complete for any rational* $r > 0$.

5.3 Complexity of k Equal Sum Subsets

5.3.1 NP-completeness of k Equal Sum Subsets

We now study the problem k Equal Sum Subsets, where we ask for k subsets of equal sum (note that Equal Sum Subsets is the special case where $k = 2$). We first show its NP-hardness by reduction from Alternating Partition (see Definition 2.4.7).

Theorem 5.3.1. k Equal Sum Subsets *is* NP*-complete for any* $k \geq 2$.

Proof: The problem is obviously in NP. NP-hardness for the case $k = 2$ follows immediately from the fact that Equal Sum Subsets is NP-complete. To show NP-hardness for $k > 2$, we reduce Alternating Partition to it. We transform a given Alternating Partition instance with pairs $(u_1, v_1), \ldots, (u_n, v_n)$ into a k Equal Sum Subsets instance as follows. For each pair (u_i, v_i) we construct two numbers $u'_i = \langle u_i \rangle \circ \Delta_n(i)$ and $v'_i = \langle v_i \rangle \circ \Delta_n(i)$. In addition, we construct $k-2$ (equal) numbers $c_1, \ldots, c_{k-2}$ with $c_i = \langle \frac{1}{2} \sum_i (u_i + v_i) \rangle \circ \mathbb{1}_n$. We set base $Z = (n+1) \cdot k \cdot \sum_i (u_i + v_i)$, which is chosen sufficiently large to ensure that no carry–overs from one digit to the next occur in any of the following additions.

Assume that we are given a solution of the Alternating Partition instance, i.e., two index sets I and J such that $\sum_{i\in I} u_i + \sum_{j\in J} v_j = \sum_{i\in I} v_i + \sum_{j\in J} u_j$. We construct k equal sum subsets $S_1, \ldots, S_k$ as follows. For $k = 1, \ldots, k-2$, we have $S_i = \{c_i\}$; for the remaining two subsets, we let $u_i' \in S_{k-1}$, if $i \in I$, and $v_i' \in S_{k-1}$, if $i \in J$, and we let $u_i' \in S_k$, if $i \in J$, and $v_i' \in S_k$, if $v_i \in I$. Obviously, all S_i sum up to the same sum $\langle \frac{1}{2}\sum_i (u_i + v_i)\rangle \circ \mathbb{1}_n$, thus we have a solution for the k Equal Sum Subsets instance.

For the opposite direction, assume that we are given a solution of the k Equal Sum Subsets instance, i.e., k equal sum subsets $S_1, \ldots, S_k$. Since each of the n right–most digits is set to 1 in exactly k numbers, we can assume w.l.o.g. that $S_i = \{c_i\}$ for $i = 1, \ldots, k-2$. The remaining two subsets naturally form an alternating partition, as u_i' and v_i' can never be in the same subset for any $i = 1, \ldots, n$, and all numbers u_i' and v_i' must occur in one of the remaining two subsets in order to match the ones in the n right–most digits of the other subsets.

□

5.3.2 k Equal Sum Subsets for $k = O(1)$ and $k = \Omega(n)$

We now study the impact of the size of parameter k on the complexity of k Equal Sum Subsets. In particular, we show that the problem can be solved in pseudo–polynomial time if k is a constant, while it becomes strongly NP-hard if k is linear in n.

Theorem 5.3.2. *The problem k Equal Sum Subsets with input $A = \{a_1, \ldots, a_n\}$ can be solved in time $O(\frac{n \cdot S^k}{k^{k-1}})$, where $S = \text{sum}(A)$. For $k = O(1)$, this time is pseudo–polynomial.*

Proof: We present a dynamic programming algorithm for k Equal Sum Subsets that uses basic ideas of well–known dynamic programming algorithms for Bin Packing with fixed number of bins [40].

For an instance $A = \{a_1, \ldots, a_n\}$ of k Equal Sum Subsets, let $S = \text{sum}(A)$. We define Boolean variables $F(i, s_1, \ldots, s_k)$, where $i \in \{1, \ldots, n\}$ and $s_j \in \{0, \ldots, \lfloor \frac{S}{k} \rfloor\}$, for $1 \le j \le k$. Variable $F(i, s_1, \ldots, s_k)$ will be TRUE if there are k disjoint subsets $X_1, \ldots, X_k \subseteq \{a_1, \ldots, a_i\}$ with $\text{sum}(X_j) = s_j$, for $1 \le j \le k$. Given this, there is a solution for the k Equal Sum Subsets instance if and only if there exists a value $s \in \{1, \ldots, \lfloor \frac{S}{k} \rfloor\}$ such that $F(n, s, \ldots, s) = \text{TRUE}$.

Clearly, $F(1, s_1, \ldots, s_k)$ is TRUE if and only if either $s_i = 0$, for $1 \leq i \leq k$, or there exists index j such that $s_j = a_1$ and $s_i = 0$, for all $1 \leq i \leq k, i \neq j$. For $i \in \{2, \ldots, n\}$ and $s_j \in \{0, \ldots, \lfloor \frac{S}{k} \rfloor\}$, variable $F(i, s_1, \ldots, s_k)$ can be expressed recursively as

$$\begin{aligned} F(i, s_1, \ldots, s_k) &= F(i-1, s_1, \ldots, s_k) \ \vee \\ & \bigvee_{\substack{1 \leq j \leq k \\ s_j - a_i \geq 0}} F(i-1, s_1, \ldots, s_{j-1}, s_j - a_i, s_{j+1}, \ldots, s_k). \end{aligned}$$

The Boolean value of all variables can be determined in time $O(\frac{nS^k}{k^{k-1}})$, since there are $n\lfloor \frac{S}{k} \rfloor^k$ variables, and computing each variable takes at most time $O(k)$. This yields the claim.

□

The previous theorem shows that there is a pseudo–polynomial time algorithm for k Equal Sum Subsets if k is a constant. We will now show that this is unlikely if k is a fixed *function* of the cardinality n of the input set. In fact, we prove that k Equal Sum Subsets is strongly NP-complete if $k = \Omega(n)$, by reduction from 3–Partition (see Definition 2.4.1). For this purpose, let $k = \frac{n}{p}$, for any arbitrary but fixed integer $p \geq 2$.

Theorem 5.3.3. *k Equal Sum Subsets is NP-complete in the strong sense for $k = \frac{n}{p}$, for any fixed integer $p \geq 2$.*

Proof: The problem k Equal Sum Subsets is obviously in NP. To prove strong NP-hardness, we reduce 3–Partition to it. Let $Q = \{q_1, \ldots, q_{3n}\}$ and h be an instance of 3–Partition. If all elements in Q are equal, then there is a trivial solution. Otherwise, let $r = 3 \cdot (p-2) + 1$ and

$$\begin{aligned} a_i &= \langle q_i \rangle \circ \mathbb{0}_r, \text{ for } 1 \leq i \leq 3n, \\ b_j &= \langle h \rangle \circ \mathbb{0}_r, \text{ for } 1 \leq j \leq 2n, \text{ and} \\ d_{\ell,m} &= \langle 0 \rangle \circ \Delta_r(\ell), \text{ for } 1 \leq \ell \leq r, 1 \leq m \leq n. \end{aligned}$$

Here, we use base $Z = 6nh$ for all numbers. Let A be the multiset that contains all numbers a_i, b_j and $d_{\ell,m}$. Multiset A is an instance of k Equal Sum Subsets. The cardinality of A is $n' = 3n + 2n + r \cdot n = 5n + (3 \cdot (p-2) + 1) \cdot n = 3pn$. Since r is a constant, the numbers a_i and b_j are polynomial in h, and numbers $d_{\ell,m}$ are bounded by a constant. We now prove that there is a solution for the 3–Partition instance if and only if there are $k = \frac{n'}{p} = 3n$ disjoint subsets of A with equal sum.

"only if": Assume that there is a solution for the 3–Partition instance, i.e., n triples $T_1, \ldots, T_n$ that each sum up to h. This induces n subsets of A with sum $\langle h \rangle \circ \mathbb{0}_r$, namely $S_k = \{a_i \mid q_i \in T_k\}$. Together with the $2n$ subsets that contain exactly one of the b_j's each, we have $3n$ subsets of equal sum $\langle h \rangle \circ \mathbb{0}_r$.

"if": Assume that there is a solution $S_1, \ldots, S_{3n}$ for the k Equal Sum Subsets instance. Recall that for our instance $k = 3n$. Let S_j be any set in this solution. Then sum (S_j) has a zero in the r right–most digits, since for each of these digits there are only n numbers in A for which this digit is non–zero, which are not enough to have one of them in each of the $3n$ sets S_j. Thus, only numbers a_i and b_j can occur in the solution; moreover, we only need to consider the first digit of these numbers, as the other are zeros.

Since not all numbers a_i are equal, and the solution consists of $\frac{n'}{q} = 3n$ disjoint sets, there must be at least one b_j in one of the subsets in the solution. Thus, for $1 \leq j \leq 3n$, we have sum $(S_j) \geq h$. On the other hand, the sum of all a_i's and of all b_j's is exactly $3n \cdot h$, therefore sum $(S_j) = h$, for all $1 \leq j \leq 3n$, which means that all a_i's and all b_j's must appear in the solution. More specifically, there must be $2n$ sets in the solution such that each of them contains exactly one of the b_j's, and each of the remaining n sets in the solution consists only of a_i's, such that the corresponding q_i's add up to h. Thus, the latter sets immediately yield a solution for the 3–Partition instance.

□

5.3.3 k Equal Sum Subsets with Equal Cardinalities

In this section, we study k Equal Sum Subsets in the setting where we do not only require the subsets to be of equal sum, but to be of equal cardinality as well. We show that the problem can be solved in polynomial time if the cardinality is part of the problem definition (hence, a constant), while it is NP-complete if the cardinality is part of the input, or not specified at all.

We first observe that the problem kESS Of Cardinality c, where we ask for k subsets of equal sum that have cardinality c, can be solved in polynomial time using exhaustive search: We simply compute all $N = \binom{n}{c}$ subsets of the input set A that have cardinality c; then we consider all $\binom{N}{k}$ possible combinations of k subsets, and for each one we check if it consists of disjoint subsets of equal sum. This algorithm needs time $O(n^{ck})$, which is polynomial in n, as c and k are constants. This yields the following.

Theorem 5.3.4. *The problem kESS Of Cardinality c can be solved in time $O(n^{ck})$.*

On the other hand, if the size of the subsets is not fixed, but given as part of the input, then we have the problem kESS Of Specified Cardinality. We now show that this problem is NP-complete, by modifying the reduction from Alternating Partition used in the proof of Theorem 5.3.1 to show NP-completeness of k Equal Sum Subsets.

Theorem 5.3.5. *kESS Of Specified Cardinality is NP-complete for any $k \geq 2$.*

Proof: The problem kESS Of Specified Cardinality is obviously in NP. To show NP-hardness, we transform a given Alternating Partition instance $(u_1, v_1), \dots, (u_n, v_n)$ into a kESS Of Specified Cardinality instance as follows. Let $S = \sum_{i=1}^{n}(u_i + v_i)$. For each pair (u_i, v_i) we construct two numbers $u'_i = \langle u_i \rangle \circ \Delta_n(i)$ and $v'_i = \langle v_i \rangle \circ \Delta_n(i)$. In addition, we construct $k-2$ (equal) numbers $b_1, \dots, b_{k-2}$ with $b_i = \langle \frac{S}{2} \rangle \circ \Delta_n(n)$. Finally, for each b_i we construct $n-1$ numbers $d_{i,j} = \langle 0 \rangle \circ \Delta_n(j)$, for $1 \leq j \leq n-1$. We set the base of the numbers to $(n+1) \cdot k \cdot S$ in order to ensure that no carry–overs from one digit to the next occur in any additions in the following proof. The set A that contains all u'_i's, v'_i's, b_i's, and d_{ij}'s, together with chosen cardinality $c := n$, is our instance of kESS Of Specified Cardinality.

Assume first that we are given a solution for the Alternating Partition instance, i.e., two index sets I and J. We construct k equal sum subsets $S_1, \dots, S_k$ as follows. For $i = 1, \dots, k-2$, we set $S_i = \{b_i, d_{i,1}, \dots, d_{i,n-1}\}$; for the remaining two subsets, we let $u'_i \in S_{k-1}$, if $i \in I$, and $v'_j \in S_{k-1}$, if $j \in J$, and we let $u'_j \in S_k$, if $j \in J$, and $v'_i \in S_k$, if $i \in I$. Clearly, all these sets have n elements, and their sum is $\langle \frac{S}{2} \rangle \circ \mathbb{1}_n$. Hence, the sets S_i yield a solution for the kESS Of Specified Cardinality instance.

For the opposite direction, assume that we are given a solution for the kESS Of Specified Cardinality instance, i.e., k equal sum subsets $S_1, \dots, S_k$ of cardinality n. In this case, all numbers participate in the sets S_i, since there are exactly $k \cdot n$ numbers in the input A. The elements in each set S_i sum up to $\langle \frac{S}{2} \rangle \circ \mathbb{1}_n$ by definition. Since the first digit of each b_i equals $\frac{S}{2}$, we may assume w.l.o.g. that for each $i \in \{1, \dots, k-2\}$, set S_i contains b_i and does not contain any number with non–zero first digit, i.e., it does not contain any u'_j or any v'_j. Then all u'_i's and v'_i's, and only these numbers, are in the remaining two subsets. This yields immediately a solution for the Alternating Partition instance, as the two subsets yield the same sum $\langle \frac{S}{2} \rangle \circ \mathbb{1}_n$, and since u'_i and v'_i can never be in the same subset, as both have the $(i+1)$–th digit non–zero.

□

Note that the above reduction works in a similar fashion for the problem kESS Of Equal Cardinality. This requires to employ a method where additional extra digits are used in order to force the equal sum subsets to include *all* augmented numbers that correspond to numbers in the Alternating Partition instance; a similar method has been used by Woeginger and Yu to establish the NP-completeness of Equal Sum Subsets (called Equal–Subset–Sum there) [93].

However, the problems kESS Of Specified Cardinality and kESS Of Equal Cardinality are not strongly NP-complete for fixed constant k, since we will now describe a dynamic programming algorithm for these two problems.

Theorem 5.3.6. *The problems kESS Of Specified Cardinality and kESS Of Equal Cardinality with input $A = \{a_1, \ldots, a_n\}$ can be solved in time $O(\frac{S^k \cdot n^{k+1}}{k^{2k-1}})$, where $S = \mathrm{sum}\,(A)$. For $k = O(1)$, this time is pseudo–polynomial.*

Proof: The algorithm is similar–in–spirit to the dynamic programming algorithm from Theorem 5.3.2. In fact, it suffices to add to our variables k more dimensions corresponding to cardinalities of the subsets. More precisely, we define Boolean variables $F(i, s_1, \ldots, s_k, c_1, \ldots, c_k)$, where $i \in \{1, \ldots, n\}$, $s_j \in \{0, \ldots, \lfloor \frac{S}{k} \rfloor\}$, for $1 \le j \le k$, and $c_j \in \{0, \ldots, \lfloor \frac{n}{k} \rfloor\}$, for $1 \le j \le k$. Variable $F(i, s_1, \ldots, s_k, c_1, \ldots, c_k)$ will be TRUE if there are k disjoint subsets $X_1, \ldots, X_k \subseteq \{a_1, \ldots, a_i\}$ with $\mathrm{sum}\,(X_j) = s_j$, such that the cardinality of X_j is c_j, for $1 \le j \le k$. There are k subsets of equal sum and equal cardinality c if and only if there exists a value $s \in \{1, \ldots, \lfloor \frac{S}{k} \rfloor\}$ such that $F(n, s, \ldots, s, c, \ldots, c) =$ TRUE. Moreover, there are k subsets of equal sum and equal (non–specified) cardinality if and only if there exists a value $s \in \{1, \ldots, \lfloor \frac{S}{k} \rfloor\}$ and a value $d \in \{1, \ldots, \lfloor \frac{n}{k} \rfloor\}$ such that $F(n, s, \ldots, s, d, \ldots, d) =$ TRUE.

Clearly, $F(1, s_1, \ldots, s_k, c_1, \ldots, c_k) =$ TRUE if and only if either $s_i = 0$ and $c_i = 0$, for $1 \le i \le k$, or there exists an index j such that $s_j = a_1, c_j = 1$, and $s_i = 0$ and $c_i = 0$ for all $1 \le i \le k$, $i \ne j$.

For $i \in \{2, \ldots, n\}$, $s_j \in \{0, \ldots, \lfloor \frac{S}{k} \rfloor\}$, and $c_j \in \{0, \ldots, \lfloor \frac{n}{k} \rfloor\}$, variable $F(i, s_1, \ldots, s_k, c_1, \ldots, c_k)$ can be expressed recursively as

$$F(i, s_1, \dots, s_k, c_1, \dots, c_k) = F(i-1, s_1, \dots, s_k, c_1, \dots, c_k) \vee \bigvee_{\substack{1 \le j \le k \\ s_j - a_i \ge 0 \\ c_j > 0}} F(i-1, s_1, \dots, s_j - a_i, \dots, s_k, c_1, \dots, c_j - 1, \dots, c_k).$$

The Boolean value of all variables can be determined in time $O(\frac{S^k \cdot n^{k+1}}{k^{2k-1}})$, since there are $n \cdot \lfloor \frac{S}{k} \rfloor^k \cdot \lfloor \frac{n}{k} \rfloor^k$ variables, and computing each variable takes at most time $O(k)$. This yields the claim.

□

5.4 Equal Sum Subsets with Additional Requirements

5.4.1 Equal Sum Subsets with Selection Conditions

We now come back to the case of *two* subsets of equal sum (instead of k), and study variations of Equal Sum Subsets where we add specific requirements that a solution must fulfill. We start with variations where the two subsets take into account some exclusions or enforcements of specific elements of the input. Afterwards we will consider variations with constraints on the cardinalities of the two subsets.

We first study the problem ESS With Exclusions, where we are additionally given an *exclusion graph* (or its complement: a *preference graph*) and ask for two subsets of equal sum that take this graph into account. Obviously, ESS With Exclusions is NP-complete, since Equal Sum Subsets is the special case where the exclusion graph is empty ($E_{ex} = \emptyset$). Here, we present a pseudo–polynomial algorithm for the problem, using a dynamic programming approach similar–in–spirit to the one used for finding two equal sum subsets (without exclusions) [7].

Theorem 5.4.1. ESS With Exclusions *can be solved in pseudo–polynomial time $O(n^2 \cdot S)$, where $S = \text{sum}(A)$.*

Proof: Let $A = \{a_1, \dots, a_n\}$ and $G_{ex} = (A, E_{ex})$ be an instance of ESS With Exclusions. We assume w.l.o.g. that the input values are ordered by size, i.e., $a_1 \le \dots \le a_n$. Let $S = \sum_{i=1}^{n} a_i$.

We define Boolean variables $F(k, t)$ for $k \in \{1, \dots, n\}$ and $t \in \{1, \dots, S\}$. Variable $F(k, t)$ will be TRUE if there exists a set $X \subseteq A$ such that $X \subseteq$

$\{a_1, \ldots, a_k\}$, $a_k \in X$, $\text{sum}(X) = t$, and X is independent in G_{ex}. For a TRUE entry $F(k,t)$, we store a corresponding set X in a second variable $X(k,t)$.

We compute the value of all variables $F(k,t)$ by iterating over t and k. The algorithm runs until it finds the smallest $t \in \{1, \ldots, S\}$ for which there are indices $k, \ell \in \{1, \ldots, n\}$ such that $F(k,t) = F(\ell,t) = \text{TRUE}$; in this case, sets $X(k,t)$ and $X(\ell,t)$ constitute a solution $\text{sum}(X(k,t)) = \text{sum}(X(\ell,t)) = t$, both sets are disjoint due to minimality of t, and both sets are independent in G_{ex}.

We initialize the variables as follows. For all $1 \leq k \leq n$, we set $F(k,t) = \text{FALSE}$, for $1 \leq t < a_k$ and for $\sum_{i=1}^{k} a_i < t \leq S$; moreover, we set $F(k, a_k) = \text{TRUE}$ and $X(k, a_k) = \{a_k\}$. Observe that these equations already define $F(1,t)$, for $1 \leq t \leq S$, and $F(k,1)$, for $1 \leq k \leq n$.

After initialization, the table entries for $k > 1$ and $a_k < t \leq \sum_{i=1}^{k} a_i$ can be computed recursively: $F(k,t)$ is TRUE if there exists an index $\ell \in \{1, \ldots, k-1\}$ such that $F(\ell, t - a_k)$ is TRUE, and such that the subset $X(\ell, t - a_k)$ remains independent in G_{ex} when adding a_k. The recursive computation is

$$F(k,t) \quad = \quad \bigvee_{\ell=1}^{k-1} [\, F(\ell, t - a_k) \ \wedge \ \forall a \in X(\ell, t - a_k),\ (a, a_k) \notin E_{ex}\,].$$

If $F(k,t)$ is set to TRUE due to $F(\ell, t - a_k)$, then we set $X(k,t) = X(\ell, t - a_k) \cup \{a_k\}$. The key observation for showing correctness is that for each $F(k,t)$ considered by the algorithm there is at most one $F(\ell, t - a_k)$ that is TRUE, for $1 \leq \ell \leq k-1$; if there were two, say ℓ_1, ℓ_2, then $X(\ell_1, t - a_k)$ and $X(\ell_2, t - a_k)$ would be a solution to the problem instance, and the algorithm would have stopped earlier – a contradiction. This means that all subsets considered are constructed in a unique way, and therefore, no information can be lost.

In order to determine the value $F(k,t)$, the algorithm considers $k-1$ table entries. As shown above, only one of them may be TRUE; for such an entry, say $F(\ell, t - a_k)$, the (at most ℓ) elements of $X(\ell, t - a_k)$ are checked to see if they exclude a_k. Hence, the computation of $F(k,t)$ takes time $O(n)$, and the total time complexity of the algorithm is $O(n^2 \cdot S)$.

□

If we do not want to exclude elements, but on the contrary, we want to ensure that a specific element of the input occurs in one of the two equal sum subsets, then this is the ESS With Enforced Element problem.

We show that this problem is NP-complete by giving a reduction from ALTERNATING PARTITION.

Theorem 5.4.2. ESS WITH ENFORCED ELEMENT *is* NP-*complete.*

Proof: The problem ESS WITH ENFORCED ELEMENT is obviously in NP. For the proof of NP-hardness, let $(u_1, v_1), \dots, (u_n, v_n)$ be an instance of ALTERNATING PARTITION. Let $S = \sum_{i=1}^{n}(u_i + v_i)$, $a_i = \langle u_i \rangle \circ \Delta_n(i)$ and $b_i = \langle v_i \rangle \circ \Delta_n(i)$, for all $1 \leq i \leq n$, and let $c = \langle \frac{S}{2} \rangle \circ \mathbb{1}_n$. For these numbers, we use base $Z = 2 \cdot S \cdot n$, which is large enough such that no carry–overs from one digit to the next occur in the following additions.

The a_i's, b_i's, and c are an instance of ESS WITH ENFORCED ELEMENT such that c, which is the last element of the input, is the enforced element. We now show that there exists a solution for the ALTERNATING PARTITION instance if and only if there exists a solution for the ESS WITH ENFORCED ELEMENT instance.

Assume that index sets I and J are a solution for the ALTERNATING PARTITION instance. Then $\sum_{i \in I} u_i + \sum_{j \in J} v_j = \sum_{i \in I} v_i + \sum_{j \in J} u_j = \frac{S}{2}$. Let $X = \{c\}$ and $Y = \{a_i \mid i \in I\} \cup \{b_j \mid j \in J\}$. Then

$$\begin{aligned}
\text{sum}\,(Y) &= \sum_{i \in I} a_i + \sum_{j \in J} b_j \\
&= \sum_{i \in I} (\langle u_i \rangle \circ \Delta_n(i)) + \sum_{j \in J} (\langle v_j \rangle \circ \Delta_n(j)) \\
&= \langle \sum_{i \in I} u_i + \sum_{j \in J} v_j \rangle \circ (\sum_{i \in I} \Delta_n(i) + \sum_{j \in J} \Delta_n(j)) \\
&= \langle \frac{S}{2} \rangle \circ \sum_{i=1}^{n} \Delta_n(i) \\
&= \langle \frac{S}{2} \rangle \circ \mathbb{1}_n \\
&= \text{sum}\,(X),
\end{aligned}$$

thus X and Y are a solution for the ESS WITH ENFORCED ELEMENT instance.

For the opposite direction, let X and Y be a solution for the ESS WITH ENFORCED ELEMENT instance with $c \in X$. All numbers in the input have $n+1$ digits. For each index $i \in \{2, \dots, n+1\}$, only three numbers, namely c, a_i and b_i, have a 1 in the i'th digit, all other numbers in the input have a 0 in the i'th digit. For each digit the sum over all elements in X and in Y yields the same result. Therefore, since $c \in X$, exactly one of a_i or

b_i can be in Y for each $1 \leq i \leq n$, and $X = \{c\}$, since any other element would add a second 1 in some digit i, which then could not be equalized by elements in Y. Summing up the first digit of all elements in Y yields exactly the first digit of c, which is $\frac{S}{2}$. Thus, $I = \{i \in \{1, \dots, n\} \mid a_i \in Y\}$ and $J = \{j \in \{1, \dots, n\} \mid b_j \in Y\}$ yields a solution for the Alternating Partition instance.

□

We now turn to the problem Alternating Equal Sum Subsets, which is the "partial" equivalent of Alternating Partition that we used in the previous proof. In Alternating Equal Sum Subsets, we are given pairs of numbers, and we require for each element that we use in one set that its partner will be in the other set. We show that the problem is NP-complete by reduction from Equal Sum Subsets.

Theorem 5.4.3. Alternating Equal Sum Subsets *is* NP*-complete.*

Proof: The problem is obviously in NP. Given an instance of Equal Sum Subsets, i.e., a set of numbers $A = \{a_1, \dots, a_n\}$, we reduce it to an instance of Alternating Equal Sum Subsets by setting $B = 2 \cdot \sum_{i=1}^{n} a_i$ and mapping each number a_i to a pair (u_i, v_i), with $u_i = B + a_i$ and $v_i = B$. Note that we use offset B since all input numbers for Alternating Equal Sum Subsets are required to be positive. Clearly, if there are disjoint sets $X, Y \subseteq A$ such that $\text{sum}(X) = \text{sum}(Y)$, then $I := \{i \mid a_i \in X\}$ and $J := \{j \mid a_j \in Y\}$ are disjoint index sets such that $\sum_{i \in I} u_i + \sum_{j \in J} v_j = \sum_{i \in I} v_i + \sum_{j \in J} u_j$. Conversely, if there is a solution for the Alternating Equal Sum Subsets instance, i.e., appropriate sets of indices I and J, then the sets $X = \{a_i \mid i \in I\}$ and $Y = \{a_j \mid j \in J\}$ form obviously a solution for the Equal Sum Subsets instance.

□

5.4.2 Equal Sum Subsets with Cardinality Constraints

As a further class of variations of Equal Sum Subsets, we study problems with constraints on the cardinalities of the two subsets of equal sum. Obviously, if we ask for two subsets of equal sum that both have cardinality c, where c is a constant, then Theorem 5.3.4 applies to the case of $k = 2$ subsets, and we have a polynomial time algorithm for this problem. On the other hand, if c is part of the input, or if we do not specify it at all, then the problem becomes NP-complete due to Theorem 5.3.5. We will now show that the problem ESS Of Different Cardinality, where we ask for two equal sum subsets of different cardinality, is NP-complete as well.

Theorem 5.4.4. ESS Of Different Cardinality *is* NP*-complete.*

Proof: The problem is obviously in NP. The proof of NP-hardness follows directly from the reduction used in the proof of Theorem 5.4.2: If c is not in one of the two sets X or Y, then the two sets will have equal cardinality, since each a_i in one set enforces the corresponding b_i to be in the other set. Since we require sets of different cardinality, c has to be in one of the two sets, say in X, and in Y exactly one of the two numbers a_i or b_i occurs, for $1 \leq i \leq n$. Hence, set X has one element and Y has n elements.

□

Observe that we can even guarantee any specified difference $d < n$ in the cardinality of the two subsets, by "blowing up" the instance we construct: Given an instance of Alternating Partition, let $M = n \cdot d \cdot 2^{n+2}$, let $a_i = \langle u_i \rangle \circ \Delta_n(i) \circ \langle \frac{M}{n} \rangle$ and $b_i = \langle v_i \rangle \circ \Delta_n(i) \circ \langle \frac{M}{n} \rangle$, for all $1 \leq i \leq n$, and let $c = \langle \frac{S}{2} \rangle \circ \mathbb{1}_n \circ \langle M - (2^{n-d-1} - 1) \rangle$. For $1 \leq k \leq n-d-1$, we define dummy elements $d_k = \langle 0 \rangle \circ \mathbb{0}_n \circ \langle 2^{k-1} \rangle$. Like in the previous proof, any solution with only a_i's and b_i's will have equal cardinality. Thus, c has to be in one of the sets, say X, and n of the a_i's and b_i's will be in the other set Y to achieve equal sums in the first $n+1$ digits of the elements in X and Y. To achieve an equal sum in the last digit as well, d_k must be in set X, for all $1 \leq k \leq n-d-1$. Hence, $|X| = n - d$ and $|Y| = n$.

5.4.3 Finding Equal Sum Subsets from Two Sets

As a last class of variations of Equal Sum Subsets, we now study the complexity of ESS From Two Sets, in which we ask for two equal sum subsets of two different sets. Obviously, ESS From Two Sets and Alternating Equal Sum Subsets are very closely related. We first show that ESS From Two Sets is NP-complete, by reducing Subset Sum to it (see Definition 2.4.6); then we show that the problem remains hard to solve even if we further restrict the set of possible solutions.

Theorem 5.4.5. ESS From Two Sets *is* NP*-complete.*

Proof: The problem is obviously in NP. For the NP-hardness proof, let $\{p_1, \ldots, p_n\}$ and S be an instance of Subset Sum. Let $A = \{p_1, \ldots, p_n\}$ and $B = \{b_1\}$, with $b_1 = S$. Then A and B are an instance of ESS From Two Sets. Obviously, any solution for this instance must sum up to S, since S is the only element in B. Thus, solutions for the Subset Sum instance transform straightforwardly into solutions for the ESS From Two Sets instance, and vice versa.

□

We now study variations of the problem where we introduce additional constraints on the indices used in a solution. These results show that NP-hardness of ESS From Two Sets is stable under decreasing the size of the solution space. In fact, we show NP-completeness for our four variations of ESS From Two Sets where we restrict the choice of indices in different ways.

Theorem 5.4.6. *The following problems are* NP*-complete:*

ESS Of Equal Cardinality From Two Sets*,*
ESS With Disjoint Indices From Two Sets*,*
ESS With Disjoint Covering Indices From Two Sets*, and*
ESS With Identical Indices From Two Sets*.*

Proof: Obviously, each of the problems is in NP. We show NP-hardness individually for each problem variation.

To prove NP-hardness of ESS Of Equal Cardinality From Two Sets, we give a reduction from Subset Sum. Given an instance $\{p_1, \dots, p_n\}$ and S of Subset Sum, we construct an instance A, B of ESS Of Equal Cardinality From Two Sets as follows. Let $a_i = \langle p_i, i, 0\rangle$, for $1 \leq i \leq n$, and $a_{n+1} = \langle 0, 0, 1\rangle$. Let $b_i = \langle 0, i, 0\rangle$, for $1 \leq i \leq n$, and $b_{n+1} = \langle S, 0, 1\rangle$. Set A consists of all a_i's, and set B of all b_i's. We now show that a solution for the Subset Sum instance yields a solution for the instance A, B of ESS Of Equal Cardinality From Two Sets, and vice versa.

If there is a set $X \subseteq \{p_1, \dots, p_n\}$ such that $\mathrm{sum}\,(X) = S$, then we set $I = \{i \mid x_i \in X\} \cup \{n+1\}$. This yields $\sum_{i \in I} a_i = \sum_{j \in I} b_j = \langle S, k, 1\rangle$, where $k = \sum_{x_i \in X} i$, thus the two subsets defined by indices I and J have equal sum and equal cardinality.

For the opposite direction, assume that two non–empty sets $I, J \subseteq \{1, \dots, n\}$ exist such that $|I| = |J|$ and $\sum_{i \in I} a_i = \sum_{j \in J} b_j$. Then $n+1 \in J$ is necessary to have equal sums in the first digit, and moreover, we must have $\sum_{i \in I} p_i = S$. Thus, the corresponding p_i's yield a solution for the Subset Sum instance.

To prove NP-hardness of ESS With Disjoint Indices From Two Sets, we give a reduction from ESS From Two Sets. Given an instance $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ of ESS From Two Sets, we can construct an instance of ESS With Disjoint Indices From Two Sets as follows. Let $a'_i = \langle a_i\rangle \circ \mathbb{0}_n \circ \mathbb{0}_n$ and $a'_{n+i} = \langle 0\rangle \circ \mathbb{0}_n \circ \Delta_n(i)$, for all $1 \leq i \leq n$, and let $b'_i = \langle 0\rangle \circ \Delta_n(i) \circ \mathbb{0}_n$ and $b'_{n+i} = \langle b_i\rangle \circ \mathbb{0}_n \circ \mathbb{0}_n$, for $1 \leq i \leq n$. Set A' consists of all a_i's, and B' consists of all b_i's. It is easy to see that there are two equal sum subsets of A and B if and only if there are equal sum subsets of A' and B' with disjoint indices, since only subsets of the first n numbers in A' and the last n numbers in B' can yield equal sums.

To prove NP-hardness of ESS With Disjoint Covering Indices From Two Sets, we give a reduction from Partition (cf. Definition 2.4.5). Given an instance $A = \{a_1, \ldots, a_n\}$ of Partition, we construct an instance of ESS With Disjoint Covering Indices From Two Sets by setting $A' = B' = A$. If A can be partitioned into subsets X and Y, then choosing the corresponding elements in A' and B', respectively, gives us a solution for the ESS With Disjoint Covering Indices From Two Sets instance, and vice versa.

Finally, we prove NP-hardness of ESS With Identical Indices From Two Sets, by using the same reduction as for ESS Of Equal Cardinality From Two Sets (see above): It suffices to observe that any two equal sum subsets $U \subseteq A$ and $V \subseteq B$ either have identical indices, or there is always $V' \subseteq B$ such that $\mathrm{sum}(V) = \mathrm{sum}(V') = \mathrm{sum}(U)$, and such that V' has identical indices with U.

□

5.5 Conclusion

We studied several variations of the Equal Sum Subsets problem: We proved NP-completeness for the variation where we specify a rational factor between the sum of the two subsets (Factor–r Sum Subsets). If we ask for more than two equal sum subsets (k Equal Sum Subsets), then the problem becomes strongly NP-hard, if the number of subsets is linear in n (the size of the input), while it can be solved in pseudo–polynomial time if we ask for only a constant number of subsets. If we require the k subsets to be of equal cardinality, then the problem is polynomial–time solvable if the cardinality is constant, while it is NP-hard otherwise. Furthermore, we proved NP-hardness for several variations of Equal Sum Subsets where the two subsets have to fulfill additional requirements, namely for ESS With Exclusions, ESS With Enforced Element, and Alternating Equal Sum Subsets. Finally, we introduced the problem where we ask for two equal sum subsets from two sets, and showed that this problem is NP-hard, even if we restrict the choices of the elements.

Although our interest in the Equal Sum Subsets problem was motivated from its connection to the Partial Digest problem, our results did not yield any new insights in the complexity of Partial Digest. However, our studies call forth several questions in the realm of Equal Sum Subsets itself that are still open:

- The problem k Equal Sum Subsets is solvable in pseudo–polynomial time for constant k, while it is strongly NP-complete for k linear in n. What is the exact borderline between pseudo–polynomial time solvability and strong NP-hardness?
- The dynamic programming algorithm for kESS Of Specified Cardinality runs in pseudo–polynomial time. However, its running time is highly exponential in k; are there faster algorithms for this problem?
- We have only studied variations where the subsets need to have exactly the same sum. What about approximation versions related to the above problems, for instance if we ask for k subsets of A with sums that are "as similar as possible"? For $k = 2$, the problem has been studied by Bazgan et al. [7] and Woeginger and Yu [93].

Chapter 6

Mass Finding in Weighted Strings

6.1 Introduction

The Mass Finding problem, where we search a weighted string for a specific submass, arises when we want to identify a protein using its mass fingerprint. We recapitulate the definition from the introduction (cf. Definition 1.3.1):

Definition. *Given an alphabet $\mathcal{A}$, a mass function $\mu : \mathcal{A} \to \mathbb{N}$, and a string σ over $\mathcal{A}$, find a data structure and a query algorithm which, for a given positive integer M, decides whether σ has a substring of mass M, where the mass of a string is the sum of the masses of its letters.*

Due to its importance in proteomics, protein identification by mass fingerprints has been extensively studied in the literature, e.g. in [33, 35, 45, 50, 66, 99, 97]. Many papers deal with specific aspects and modifications of the problem, e.g. the minimum number of matches needed to identify a protein [66], combinatorial or probabilistic models for scoring the differences of two mass spectra [4, 72], or approaches for a correct identification even in the presence of post–translational modifications of the protein [59, 73, 98]. There are several software tools for automated database search, for instance Sequest [33, 112], Mascot [69, 108], or Sonar [37].

In this chapter, we study the algorithmic complexity of the Mass Finding problem. The Mass Finding problem differs from traditional string searching problems in one important aspect: While the latter look for substructures of strings (substrings, non–contiguous subsequences, particular types of substrings such as repeats, palindromes etc.), we are interested only in masses respectively weights of substrings. This means that, on the one hand, we lose a lot of the structure of strings: for instance, the weight of a string is invariant under permutation of letters; on the other, we gain the additional structure of the weight function, such as its additivity. For instance, using suffix trees, which can be applied to efficiently solve a large number of complex string problems, do not seem to help for the Mass Finding problem. Furthermore, the longest common substring problem [43], although at first sight related, has very different characteristics. A problem that may also appear to be close to the present one is maximum segment sum [9]; however, it appears that it does not lead to good solutions, either. In fact, we are not aware of any results related to efficient algorithms for the Mass Finding problem.

In Section 6.2, we first fix some notation, and then we present simple algorithms that solve the Mass Finding problem. We first show that we can answer queries in linear time even without using any additional data structure, where time (and space) complexity is measured in the length of the input string. On the other hand, if we allow preprocessing, then we can use a hash table or a sorted array to store all submasses that occur in the string (recall that a submass is the mass of a contiguous substring). This yields constant respectively logarithmic query times. The space required by both data structures depends on the number of different submasses of the string, and can be up to quadratic. In the special case that we know in advance that we will always search for short masses, i.e., for masses that arise from only few amino acids, we can reduce the amount of data stored in the hash table respectively binary array and obtain a smaller data structure. However, in the general case we may need quadratic space for the data structures.

We then consider the generalization where we want to search for mass M in not only one string, but in many strings simultaneously. This problem is defined as follows.

Definition 6.1.1 (Multiple–String Mass Finding). *Given an alphabet $\mathcal{A}$, a mass function $\mu : \mathcal{A} \to \mathbb{N}$, and k strings $\sigma_1, \ldots, \sigma_k$ over $\mathcal{A}$, find a data structure and a query algorithm which, for a given mass $M \in \mathbb{N}$, returns a list $i_1, \ldots, i_r$ of those strings σ_{i_j} that have M as a submass.*

Obviously, we can run an algorithm for the Mass Finding problem individually for each string. We show in Section 6.2.3 that we can use a kind of "binary search technique" to search many strings simultaneously, thus allowing to solve the Multiple–String Mass Finding problem more efficiently.

We then come back to the case of only one string. Motivated by the simple algorithms for the Mass Finding problem, we ask for algorithms for the problem that allow for sublinear query times and that use only subquadratic additional storage space. We design in Section 6.3 an algorithm called Lookup that meets these efficiency requirements, as it needs only linear additional storage space and has sublinear query time. However, the algorithm requires unreasonably large inputs to become efficient; hence, our result is primarily of theoretical impact, as it proves that both sublinear time and subquadratic space can be achieved at the same time.

Observe that any algorithm of practical value, in addition to being time and space efficient, also needs to be highly fault tolerant. In fact, in real life all MS/MS data are prone to error, and for practical applications the Mass Finding problem should be relaxed to finding a submass that matches the given mass M up to some error ε. However, we do not address this relaxation here.

The results in this chapter have been published previously [24, 25, 26].

6.2 Simple Solutions

6.2.1 Notation

Fix an *alphabet* $\mathcal{A}$ of size $|\mathcal{A}| = s$ and a *mass function* $\mu : \mathcal{A} \to \mathbb{N}$. Let $\sigma = \sigma(1) \dots \sigma(n)$ be a string over $\mathcal{A}$ of length $|\sigma| = n \geq 1$. We denote by $\sigma(i, j)$, for i, j with $1 \leq i \leq j \leq n$, the substring of σ starting at position i and ending at position j, i.e., $\sigma(i, j) = \sigma(i) \dots \sigma(j)$. A non–empty string τ is a substring of σ if there are $1 \leq i \leq j \leq n$ such that $\tau = \sigma(i, j)$. Note that we do not consider the empty string to be a substring. The *mass* (or *weight*) of σ is defined as the sum of the individual masses of its letters: $\mu(\sigma) := \sum_{i=1}^{n} \mu(\sigma(i))$. For a mass $M \in \mathbb{N}$, we say that M is a *submass* of σ if σ has a substring of mass M.

6.2.2 Simple Algorithms for Mass Finding

In this section, we present several simple algorithms that solve the Mass Finding problem.

A first algorithm is Linsearch, which performs a linear search through the string. This algorithm can be visualized as shifting two pointers ℓ and r through the string, where ℓ points to the beginning of a substring and r to its end. Linsearch works as follows. For given σ and M, start at position $\sigma(1)$ and add up masses until reaching the first position j such that $\mu(\sigma(1, j)) \geq M$. If the mass of the substring $\sigma(1, j)$ equals M, then we have found M, thus we output **yes** and stop; otherwise, start subtracting masses from the beginning of the string until the smallest index i such that $\mu(\sigma(i,j)) \leq M$ is reached. Repeat this until finding a pair of indices (i,j) such that $\mu(\sigma(i,j)) = M$, or until reaching the end of the string, i.e., until the current substring is $\sigma(i,n)$, for some i, and $\mu(\sigma(i,n)) < M$. LINSEARCH takes $O(n)$ time, since it looks at each position of σ at most twice. If we do not allow any preprocessing, this is asymptotically optimal, since it may be necessary to look at each position of σ at least once.

On the other hand, if preprocessing of σ is allowed, then there is another simple algorithm for the MASS FINDING problem, which uses binary search: In a preprocessing step, it calculates the set of all possible submasses of σ, i.e., $\mu(\sigma(i,j))$, for all $1 \leq i \leq j \leq n$, and stores them in a sorted array. Given a query mass M, it performs binary search for M in this array. We will refer to this algorithm as BINSEARCH. The space required to store the sorted array is proportional to the number of different submasses in σ, which is bounded by $O(n^2)$. The time for answering a query is thus $O(\log n)$.

Since our submasses are integers, we can use a hash table instead of a sorted array to store all submasses of σ. In fact, there exist hashing schemes that require storage space linear in the number of elements to be stored, and which allow membership queries in constant time [38]. For the MASS FINDING problem, this yields an algorithm that we refer to as HASHING with space proportional to the number of different submasses in σ, and constant query time.

Observe that the problem becomes easier if we assume that all masses we are looking for will be "short masses": For a mass M we define its *length* as $\lambda(M) := \max(\{|\tau| \mid \tau \in \mathcal{A}^*, \mu(\tau) = M\} \cup \{-1\})$. Here, $\lambda(M) = -1$ means that there exists no string with mass M. Suppose that we know in advance that *all* query masses will be short in comparison to n, i.e., that there is a function $f(n) = o(n)$ such that $\lambda(M) \leq f(n)$ for all queries M. Then we can improve the space required by BINSEARCH as follows: In the preprocessing, we store only submasses of σ of length $\ell \leq f(n)$ in the sorted array. This requires storage space $O(n \cdot f(n))$, since for each position i in σ, at most $f(n)$ substrings of length $\ell \leq f(n)$ start in i. For a query, we do binary search in this array. This takes still time $O(\log n)$, while the storage space required by the array is subquadratic. Observe that we can improve HASHING in a similar fashion. Moreover, we can use this approach as a first phase of any algorithm to make it run faster on short masses.

Another simple algorithm for the MASS FINDING problem, which we will refer to as BOOLEANARRAY, works as follows. In the preprocessing phase, define $W := \max\{\mu(a) \mid a \in \mathcal{A}\}$, and let B be a Boolean array of length $\mu(\sigma)$. Set $B[k]$ to TRUE if and only if k is a submass of σ. Given a query mass M, we simply output $B[M]$. This algorithm has query time

$O(1)$, while the data structure B requires $\mu(\sigma) \leq n \cdot W$ bits. Thus, the algorithm is very efficient if $W = o(n)$. However, this does not solve the Mass Finding problem in general, since we do not want to restrict the size of W.

In the following, we assume that the alphabet $\mathcal{A}$ is of constant size and we do not restrict the maximum letter weight W. We assume a machine model with word size $S := \Omega(\log n + \log W)$ in which arithmetic operations on numbers with S bits can be executed in constant time; storage space is measured in terms of the number of machine words used. Without this assumption, we would get an extra factor S in the query time and in the storage space. Since the alphabet is of constant size, an input string σ of length n could be stored in $O(\frac{n}{S})$ machine words. However, we will assume that the input string occupies n machine words.

6.2.3 Algorithms for Multiple–String Mass Finding

We now study how we can search several strings simultaneously. Obviously, any algorithm Ψ for the Mass Finding problem can be extended to an algorithm for the Multiple–String Mass Finding problem by running Ψ on each string σ_i one by one. Required storage space and query time simply sum up. Alternatively, we can adapt an approach from group testing [32]: We define a new string $\sigma := \sigma_1\omega\sigma_2\omega\ldots\omega\sigma_k$, where ω is a new letter with mass $\mu(\omega) := \max\{\mu(\sigma_i) \mid 1 \leq i \leq k\} + 1$, i.e., the mass of ω is larger that any submass of the given strings. Before applying Ψ to σ, we check whether $M \geq \mu(\omega)$. If so, then M cannot be a submass of any of the strings, and we are done. Otherwise, we know that whenever Ψ finds mass M in σ, then M is a submass of at least one σ_i, for some index i, as the corresponding substring of σ cannot contain letter ω. If algorithm Ψ can output *all* positions of M in σ, this solves the Multiple–String Mass Finding problem. If Ψ only *decides* whether M is a submass of σ, i.e., it outputs only **yes** or **no**, then we employ algorithm BinTreeSearch, which uses a kind of "binary tree search" to find all σ_i with submass M as follows. First, we run Ψ on σ as described above. If it outputs **no**, then no string σ_i has submass M, and we are done. Otherwise, we divide σ into two new strings $\sigma_l := \sigma_1\omega\ldots\omega\sigma_{\lfloor\frac{k}{2}\rfloor}$ and $\sigma_r := \sigma_{\lfloor\frac{k}{2}\rfloor+1}\omega\ldots\omega\sigma_k$, and run Ψ on both strings separately. We repeat the division step until the new strings cover exactly one σ_i, in which case the answer of Ψ determines whether σ_i has a substring of mass M. The analysis of BinTreeSearch depends heavily on storage space and query time required by Ψ. For instance, if algorithm Ψ requires storage space linear in the length of the string, then the storage space of BinTreeSearch is $O((\log k) \cdot \sum_{i=1}^{k} |\sigma_i|)$. Query time of BinTreeSearch depends on the number of strings with submass M, i.e., it is output sensitive, in contrast to the simple idea of applying Ψ to each string separately, which depends on the total number of strings.

Given a specific algorithm for the Mass Finding problem, there can be even better ways to extend it to the Multiple–String Mass Finding problem: For instance, for algorithm Binsearch we can use *one* sorted array to store all submasses of all strings. For each submass x we store the set of indices I_x of all those strings that have a submass x. Given mass M, we perform binary search in the array and output all indices stored in I_M. Required storage space remains unchanged, but the running time becomes $O(\log(\sum_{i=1}^{k} |\sigma_i|) + |I_M|)$, where $|I_M| \leq k$ is the size of the output.

6.2.4 Efficiency Measurements

An algorithm for the Mass Finding problem can be divided into three components: A preprocessing phase, a data structure in which the result of the preprocessing is stored, and a query method. For a string σ, the preprocessing will be done only once, for instance, when the protein enters the database, while the query step will typically be repeated many times. For this reason, we are interested in algorithms with fast query methods, whereas we ignore time and space required for the preprocessing step (as long as they are within reasonable bounds). Space efficiency is measured in storage space required by the data structure.

The two algorithms Binsearch and Hashing are very efficient in query time; however, they both can require up to quadratic storage space, which can be immense for long strings σ. In the other extreme, Linsearch requires no additional storage space, but its query time is only linear in the string length, which is slow for large databases. For this reason, we are looking for algorithms that have query time better than Linsearch, i.e., that require query time $o(n)$, to allow for fast database search, while they only require little additional storage space, i.e., only storage space $o(n^2)$ for the data structure.

6.3 Algorithm Lookup

We now present an algorithm called Lookup that solves the Mass Finding problem with storage space $O(n)$ and query time $O(\frac{n}{\log n})$. The idea is as follows. Similar to the simple linear search algorithm Linsearch that we introduced in Section 6.2, Lookup shifts two pointers along the sequence which point to the potential beginning and end of a substring with mass M. However, $c(n)$ steps of the simple algorithm are bundled into one step here.

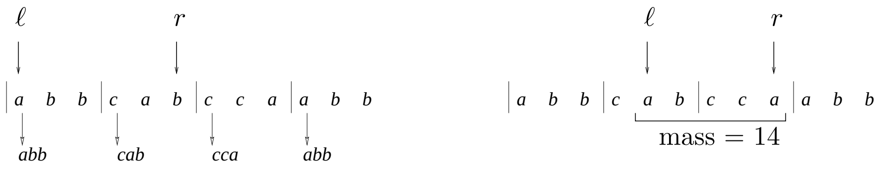


Figure 6.1: Example for algorithm LOOKUP searching for $M = 14$.

This will reduce the number of steps from $O(n)$ to $O(\frac{n}{c(n)})$, while each step will still require only constant time. If $c(n)$ is chosen appropriately, i.e., approximately $\log n$, then the storage space required will be $O(n)$. We will hereby heavily exploit the fact that the alphabet has constant size.

An Example

Before we present the complete algorithm, we explain its main ideas in an example. Let $\mathcal{A} = \{a, b, c\}, \mu(a) = 1, \mu(b) = 2$ and $\mu(c) = 5$. Assume that we are looking for $M = 14$ in string $\sigma = abbcabccaabb$. LINSEARCH would pass two pointers ℓ and r along the sequence until reaching positions 5 and 9, respectively. Here, it would stop because the substring $\sigma(5, 9) = abcca$ has weight 14. To see how LOOKUP works, let us assume that $c(n) = 3$. We divide the sequence σ into blocks of size $c(n)$. Now, rather than shifting the two pointers letter by letter, we shift them by a complete block at a time. In order to do this, for each block we store a pointer to an index I corresponding to the substring that starts with the first letter of the block and ends with the last. Let us assume now that ℓ is at the beginning of the first block, and r is at the end of the second block, as indicated in Figure 6.1. We are interested in the possible *changes* to the current submass if we shift the two pointers at most $c(n)$ positions to the right. Given a list of these, we could search for $M - \mu(\sigma(\ell, r))$. For example, the current submass in Figure 6.1 is $\mu(\sigma(1, 6)) = 13$, and we want to know whether, by moving ℓ and r at most 3 positions to the right, we can achieve a change of $14-13 = 1$.

We can calculate these possible changes and store them in a $(c(n)+1) \times (c(n) + 1)$ matrix whose (i, j)–entry holds the submass change when ℓ is moved $i - 1$ positions to the right, and r is moved $j - 1$ positions to the right:

$$T[abb, cca] \quad : \quad \begin{pmatrix} 0 & 5 & 10 & 11 \\ -1 & 4 & 9 & 10 \\ -3 & 2 & 7 & 8 \\ -5 & 0 & 5 & 6 \end{pmatrix}$$

Using this matrix, we can find out whether the difference we are looking for is there, in this case +1. We will store the entries of the matrix in a hash table that will allow us to make this decision in constant time. In the present case, 1 is not in the matrix, which tells us that we have to move one of the two pointers to the next block.

To determine which pointer to move, we consider what the linear search algorithm LINSEARCH would do when searching for M and starting in the current positions of the left and right pointer. Since M is not present within these two blocks, at least one of the two pointers would reach the end of its current block. Here, we want to move the pointer which would *first* reach the end of its block. We can determine which pointer this is if we compare the difference $M-\mu(\sigma(\ell,r))$ with the matrix entry corresponding to $c(n)-1$ moves of both the left and the right pointer, in this case 7. If the difference is smaller, we move the left pointer to the next block, otherwise we move the right one. In our example, we have a difference of 1, thus we move the left pointer to the next block.

To see that this elects the correct pointer, assume by contradiction that in LINSEARCH the right pointer r would be the first to move $c(n)$ positions. Consider the "last" step, when the right pointer makes its $c(n)$'s move: previously, r has moved exactly $c(n)-1$ times, while pointer ℓ has moved at most $c(n)-1$ steps. Since LINSEARCH moves r in this configuration, the submass between ℓ and r needs to be less than M. However, this cannot be the case, since initially the submass between ℓ and r was 13, and the minimum difference we can achieve by moving the right pointer exactly $c(n)-1$ positions and the left pointer at most $c(n)-1$ positions is 7, the corresponding entry in the matrix, hence, the submass before the last step is larger than M, in contradiction to our assumption. Obviously, moving the left pointer *less* than $c(n)-1$ positions would make it even worse.

We come back to our example. Moving the left pointer will change the current submass by -5, the entry at the lower left corner of the matrix, which is in fact the minimum of the matrix as well, yielding $\mu(\sigma(4,6)) = 13-5=8$. Thus, we now look for difference $M-\mu(\sigma(4,6)) = 14-8=6$. The matrix for this pair of positions is as follows:

$$T[cab, cca] \quad : \quad \begin{pmatrix} 0 & 5 & 10 & 11 \\ -5 & 0 & 5 & 6 \\ -6 & -1 & 4 & 5 \\ -8 & -3 & 2 & 3 \end{pmatrix}$$

Value 6 is in the matrix: By looking in the matrix, we can see that a difference of 6 can be achieved by moving the left pointer by 1 position and the right pointer by 3 positions. The algorithm outputs positions 5 and 9, and then it terminates.

The Algorithm

We postpone the exact choice of the function $c(n)$ to the analysis, but assume for now that it is approximately $\log n$. For simplicity, we assume that $c(n)$ is a divisor of n.

Preprocessing: Given σ of length n, first compute $c(n)$. Next, build a table T of size $|\mathcal{A}|^{c(n)} \times |\mathcal{A}|^{c(n)}$. Each row and each column of T is indexed by a string from $\mathcal{A}^{c(n)}$. For $I, J \in \mathcal{A}^{c(n)}$, the table entry $T[I, J]$ contains the matrix of all differences $\mu(\text{prefix}(J)) - \mu(\text{prefix}(I))$ as described above. Furthermore, we store a hash table in entry $T[I, J]$ which contains the set of all entries of the matrix. Note that the table T depends only on n and $\mathcal{A}$, and not on the string σ itself. Next, divide σ into blocks of length $c(n)$. For each block, store a pointer to an index I that will be used to look up table T. Each such index I represents one string from $\mathcal{A}^{c(n)}$.

Query Algorithm: Given M, let $\ell = 1$ and $r = 0$. Repeat the following steps until M has been found or $r > n$:

1. Assume that ℓ is set to the beginning of the i'th block and r to the end of the $(j-1)$'th block. The pointer of block i resp. j points to index I resp. J. Use the hash table stored in $T[I, J]$ to find whether difference $M - \mu(\sigma(\ell, r))$ is in the corresponding matrix, i.e., whether the difference can be achieved by moving ℓ respectively r at most $c(n)$ positions to the right.

2. If the difference of $M - \mu(\sigma(\ell, r))$ can be found in the hash table, then search for an entry (k, l) in the matrix stored in $T(I, J)$ that equals $M - \mu(\sigma(\ell, r))$ by exhaustive search[1], and return **yes**, along with the witness $i' := (i-1) \cdot c(n) + k$ and $j' := (j-1) \cdot c(n) + (l-1)$, since $\mu(\sigma(i', j'))$ has mass M.

3. Otherwise, difference $M - \mu(\sigma(\ell, r))$ is not in the matrix. If $M - \mu(\sigma(\ell, r))$ is less than the matrix entry at position $(c(n), c(n))$, then increment ℓ by $c(n)$ and set $\mu(\sigma(\ell, r)) := \mu(\sigma(\ell, r)) + \min(\text{hash table})$; otherwise, increment r by $c(n)$ and set $\mu(\sigma(\ell, r)) := \mu(\sigma(\ell, r)) + \max(\text{hash table})$.

Analysis: First we derive formulas for space and time, and then we show how to choose $c(n)$. To store one entry of table T, we have to store a matrix with $(c(n) + 1)^2$ differences, and the corresponding hash table. We use a

[1]Alternatively, we could have stored (k, l) during the preprocessing, too.

hashing scheme which requires space $O(c(n)^2)$ and which allows membership queries in constant time. Such hashing schemes exist for a finite universe U of integers, see e.g. [38].

The space needed for storing the entire table T is

$$\begin{aligned}&(\text{number of entries in } T)\cdot O(c(n)^2)\\ &= |\mathcal{A}|^{2c(n)}\cdot O(c(n)^2)\\ &= O(|\mathcal{A}|^{2c(n)}\cdot c(n)^2).\end{aligned}$$

The space needed for storing the pointer at each block is

$$\begin{aligned}&\text{number of blocks}\cdot\log(\text{number of elements in } \mathcal{A}^{c(n)})\\ &= \frac{n}{c(n)}\cdot\log(|\mathcal{A}|^{c(n)}) \quad = O(n).\end{aligned}$$

For the last equality, recall that $\mathcal{A}$ is of constant size. For the query time, observe that after each iteration, consisting of Steps 1 to 3, either ℓ or r is advanced to the next block. As each of the pointers can advance at most $\frac{n}{c(n)}$ times, there can be at most $2\frac{n}{c(n)}$ iterations. Each iteration except the last one takes constant time. The last iteration may take time $O(c(n)^2)$.

In total, the algorithm requires storage space $O(n+|\mathcal{A}|^{2c(n)}\cdot c(n)^2)$ and time $O(\frac{n}{c(n)}+c(n)^2)$. Now, if we choose $c(n)=\frac{\log_{|\mathcal{A}|} n}{4}$, then we obtain $|\mathcal{A}|^{c(n)}=n^{\frac{1}{4}}$. This yields a storage space of $O(n+n^{\frac{1}{2}}\cdot\log^2 n)=O(n)$ and query time $O(\frac{n}{\log n})$. Other choices of $c(n)$ do not asymptotically improve time and space at the same time.

Theorem 6.3.1. *Algorithm* Lookup *solves the* Mass Finding *problem with storage space $O(n)$ and query time $O(\frac{n}{\log n})$.*

In principle, algorithm Lookup can be modified to work on real weights rather than on integers. Here, instead of storing the distances in hash tables, we can use sorted arrays. Each membership query to a hash table is replaced by binary search in the corresponding array. Since each array has size $O(c(n)^2)$, this results in an additional factor $O(\log c(n))$ in the query time. With $c(n)$ chosen as above, this yields storage space $O(n)$ and query time $O(\frac{n}{\log n}\log\log n)$.

Asymptotically, Lookup beats both the query time of Linsearch and the storage space of Binsearch. However, its primary purpose is to prove that such algorithms exist, since it is only efficient for very long strings: In order to obtain a block size of, say, $c(n)=10$, the input string would need to have length $n=|\mathcal{A}|^{40}$.

6.4 Conclusion

With Lookup, we presented an algorithm for the Mass Finding problem that is efficient in theory, as it requires only linear additional storage space to allow for queries in sublinear time $O(\frac{n}{\log n})$. This proves that it is asymptotically possible to beat both Linsearch and Binsearch at the same time. However, Lookup requires unreasonably large inputs to perform well. This raises the question whether there are more practical algorithms that are efficient. In the long run, it would be interesting to explore the tradeoff between query time and storage space for the Mass Finding problem. Do algorithms exist that can be parameterized to allow for an adjustment of this tradeoff?

Chapter 7

De Novo Peptide Sequencing

7.1 Introduction

De novo peptide sequencing, where we want to determine the amino acid sequence of a protein fragment from a tandem mass spectrum, is one of the most challenging problems in proteomics, and reliable techniques for automatic peptide identification are required. Several (more or less efficient) algorithms have been proposed to generate the set of matching peptide sequences for a given MS/MS spectrum; for instance, it is possible to transform a spectrum into a graph in which every connected path represents a possible sequence. Then different techniques can be used to select well matching sequences among the large number of possible paths [29, 36, 89]. Moreover, several different scoring schemes have been proposed that compare a peptide sequence to an MS/MS spectrum [53, 72, 89]. There are several software packages that implement de novo sequencing algorithms, such as Lutefisk [88, 107], BioAnalyst [103], and others [104, 105, 109]. However, the development of software tools that can generate a correct sequence for any input spectrum of reasonable quality remains an open challenge.

Recently, Chen et al. introduced a de novo sequencing algorithm that uses dynamic programming to efficiently generate all sequence candidates for a given spectrum [14]. This algorithm first transforms the given MS/MS spectrum into a graph, and then searches for a path of maximum length in this graph. Chen et al. prove that this algorithm has running time at most quadratic in the number of peaks in the given spectrum. Furthermore, they present a modification of their algorithm that allows for edge scoring functions, and that finds a maximum score path. This algorithm has cubic running time. While this approach can handle some kinds of additional peaks in the data, like single water losses, it does not work in its pure form for real–life spectra, since here the amount of noise peaks, that do not correspond to peptide ions, can be tremendeous. In fact, in typical spectra there are $100 - 400$ peaks, while the corresponding peptides consist typically of only $8 - 20$ amino acids, thus only a very small fraction of all peaks correspond directly[1] to b–ions (prefixes) and y–ions (suffixes). Noise peaks are also called *grass* peaks, since these peaks have often small abundance, hence they look like grass in the graphical representation of a spectrum (see Figure 1.7), while peaks that correspond to a peptide ion are referred to as *true* peaks.

We can use several criteria to determine how likely it is that a peak p in a spectrum corresponds indeed to a peptide ion. For instance, we can search for isotope patterns: The isotopes of an atom differ in the number of neutrons they have in the nucleus, and they occur in nature with different probabilities. E.g. carbon has either 6 neutrons, with probability 98.892%, or 7 neutrons, with probability 1.108%. If a peak p corresponds to a monoisotopic ion, then we can expect to find peak $p + 1$, and sometimes even $p + 2$, in the spectrum as well. A second criterion for a true peak can be its abundance: If p is a true peak, then p is typically one of the highest peaks in its close environment. Therefore, if the abundance of p is rather low in comparison to its surroundings, then it is unlikely that p is a true peak, although it is of course possible.

In the following, we present and discuss the prototype of a de novo sequencing tool that is based on grass mowing and the algorithm by Chen et al. This prototype is called Audens, an acronym for **AU**tomatic **DE N**ovo **S**equencing, and works as follows. The input for Audens is a `.dta`–file, which is an ASCII–formatted file that is generated from mass spectrometer raw files, for instance by Sequest [33, 112]. This `.dta`–file specifies the parent mass of the peptide, i.e., the total mass of the peptide in use, its charge state, and all peaks that occur in the spectum, i.e., masses and their corresponding abundance. An example of a `.dta` file is shown in Figure 1.5 on page 11, and Figure 1.6 shows the corresponding graphical representation.

In a first step, Audens applies a set of grass mowers to the input data, assigning to each input peak i a relevance value $r(i)$, with the default being $r(i) = 1$. We will describe the mowers in Section 7.2. Each single mower outputs values between 0 and 1, and thus, their output can be weighted against each other by using appropriate factors that can be set by the user.

[1]Some peaks in a spectrum can correspond indirectly to peptide ions, for instance if an ion loses a water or ammonia molecule.

The relevance of a peak is then the weighted sum of the values assigned by each mower. Moreover, the relevance of a solution, i.e., a peptide sequence that is supposed to match the spectrum, is the sum of the relevances of the peaks corresponding to the sequence.

In a second step, Audens applies a modification of the sequencing algorithm by Chen et al. This algorithm, that we call DYNAMIC, generates a table that – implicitly – contains all sequences that match the given spectrum. We describe DYNAMIC in Section 7.3.

Finally, Audens outputs a list of *multi–sequences* that match the input spectrum, where a multi–sequence represents a finite set of sequences that cannot be distinguished from the spectrum data. E.g. multi–sequence `V(N|GG)GYSE(I|L)ER` is short for the set {`VNGYSEIER, VNGYSELER, VGGGYSEIER, VGGGYSELER`}. The multi–sequences are ranked according to their relevances, starting with the multi–sequence that has maximum relevance. Hereby, the minimum relevance of a solution in the output can be specified by a threshold that is relative to the relevance $r_{\max}$ of a best solution; e.g. only sequences with relevance greater than 95% of $r_{\max}$ are shown. This restricts the size of the output and the running time, as described in Section 7.3.

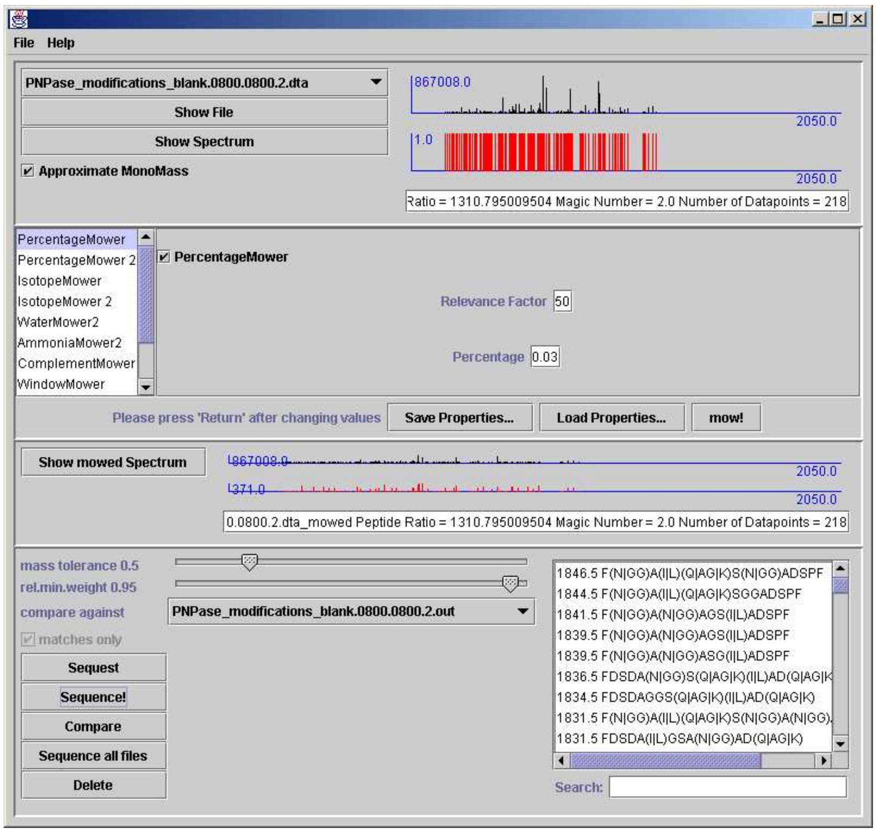
Audens comes with a graphical user interface that allows the user to select the input spectrum, set the parameters of the mowers, display the spectrum and the sequencing result in text form or in a graphical representation, and to compare the results against the output of other sequencing tools like Sequest. A screenshot is shown in Figure 7.1. We refrain from presenting the details of the implementation of Audens, since it only serves as a tool to evaluate the potential of grass mowing in de novo sequencing applications.

We have measured the performance of Audens on a test set of MS/MS spectra for which the correct peptide sequences are known. This test set, which was generated by Grossmann during his diploma thesis [42], contains 266 spectra, and Audens lists the correct peptide sequence for 79 of these spectra among its first three candidates. We discuss these results in Section 7.4, and conclude in Section 7.5 with an outlook on future developments of Audens.

Part of the results in this chapter have been published previously [6].

7.2 Grass Mowers

In this section, we present the mowers that we apply in Audens to determine the relevances of the spectrum peaks.



```
PNPase_modifications_blank.0800.0800.2.dta
867008.0   2050.0
1.0   2050.0
[✓] Approximate MonoMass
Ratio = 1310.795009504 Magic Number = 2.0 Number of Datapoints = 218

[✓] PercentageMower
Relevance Factor 50
Percentage 0.03

867008.0   2050.0
371.0   2050.0
0.0800.2.dta_mowed Peptide Ratio = 1310.795009504 Magic Number = 2.0 Number of Datapoints = 218

mass tolerance 0.5
rel.min.weight 0.95
compare against  PNPase_modifications_blank.0800.0800.2.out
[✓] matches only

1846.5 F(N|GG)A(I|L)(Q|AG|K)S(N|GG)ADSPF
1844.5 F(N|GG)A(I|L)(Q|AG|K)SGGADSPF
1841.5 F(N|GG)A(N|GG)AGS(I|L)ADSPF
1839.5 F(N|GG)A(N|GG)AGS(I|L)ADSPF
1839.5 F(N|GG)A(N|GG)ASG(I|L)ADSPF
1836.5 FDSDA(N|GG)S(Q|AG|K)(I|L)AD(Q|AG|K
1834.5 FDSDAGGS(Q|AG|K)(I|L)AD(Q|AG|K)
1831.5 F(N|GG)A(I|L)(Q|AG|K)S(N|GG)A(N|GG)
1831.5 FDSDA(I|L)GSA(N|GG)AD(Q|AG|K)
```

Figure 7.1: Screenshot of Audens.

Percentage Mower

The *percentage mower* increases the relevances of all peaks that have abundance greater than a given threshold t. Rather than taking an absolute threshold, we specify threshold t *relative* to the maximum abundance in the spectrum, e.g. 0.03% of the maximum. By this, the threshold is robust towards different intensities of measurements in different spectra.

Isotope Mower

Typically, single peptide ions give rise to more than one peak in an MS/MS spectrum due to isotopes. Thus, peaks without corresponding isotope peaks are rather unlikely to be caused by ions, and the number of isotope peaks of a single peak can be used to adjust its relevance. The *isotope mower* has a parameter k, which is the number of isotopes required to increase the relevance of a peak. Typical values for k are 1 or 2.

Water and Ammonia Mower

It can happen that a peptide ion loses a single water molecule. Thus, if p is a true peak in a spectrum, we can expect a peak at $p - 18$ as well, as 18 is the molecular mass of a water molecule. The *water mower* makes use of this fact and increases the relevance for those peaks for whom a peak at offset -18 is present as well.

The *ammonia mower* is analogous to the water mower with offset -17, as ions can lose ammonia molecules as well.

Window Mower

The *window mower* has two parameters: The size of a window w and a number k of peaks per window. It is based on the observation that, typically, high peaks are more likely to be caused by peptide ions than low peaks (this observation is used in the percentage mower as well). The window mower "slides" a window of width w over the spectrum, and for each position of the window it increases the relevances of the k peaks with the highest abundances within the window. The rationale for the window mower is twofold: First, within any window of size less than 57Da, the approximate size of the smallest amino acid mass, there can be at most two true peaks, namely one from a b–ion and one from a y–ion. Second, in many spectra contiguous regions of masses can be found such that the abundance of the peaks within *one* region do not differ very much, while they do differ between *different* regions[2]

Complement Mower

If p is a peak in the spectrum that arose from a b–ion (a prefix of the sequence), then we expect the corresponding y–ion (the corresponding suffix) to be present in the spectrum as well, and vice versa. The two masses of the two complementary peaks, plus offset $+1$ for the additional charge of the doubly charged parent ion, add up to the parent mass of the peptide. Therefore, the *complement mower* increases the relevance of a peak p if

[2]When manually sequencing, such regions are scaled with different factors in order to level the height of the peaks, and then peaks that are high within their region are considered for the sequencing process [86].

the complementary peak is present in the spectrum as well. This mower is very closely related to the sequencing algorithm we use, since this algorithm heavily relies on pairs of complementary peaks (see Section 7.3).

Second Level Mowers

For each of the mowers described above we introduce a second level variation of the mower that does not only increase the relevance of a peak p, but also the relevance of the complementary peak w.r.t. the parent mass of the peptide. These variants are referred to as *Mowername*2, where *Mowername* is any of the mowers above. This combination of a peak with its complement is – like the complement mower itself – motivated by the fact that our sequencing algorithm DYNAMIC is based on complementing pairs.

Egsit Mower

Using the mowers above we expect to assign high relevances to the true peaks in a spectrum, while grass peaks should have rather low relevance. In a final step of the preprocessing, the *egsit mower* reduces the size of the spectrum, and thus its complexity, by eliminating all peaks that are not among the x peaks with high relevances, for some parameter x. In our setting, value x is between 30 and 60, as typical peptide lengths' vary between $8-20$ amino acids, thus yielding at most $16-40$ true peaks in a spectrum.

7.3 Algorithm DYNAMIC

The sequencing algorithm DYNAMIC that we use in Audens is based on the algorithm introduced by Chen et al. [14]. The original algorithm maximizes the sum of weights of peak pairs (edges), while our algorithm maximizes the sum of the relevances assigned to the peaks. While our algorithm differs only slightly from the algorithm in [14], we present it here in order to make this thesis self–contained.

The idea of DYNAMIC is to generate a directed vertex–labeled graph $G = (V, E)$ with two special vertices x_0 and y_0, such that any directed path from x_0 to y_0 that satisfies an additional constraint will correspond to a solution, i.e., a matching sequence. For each peak i there are two vertices $x_i, y_i \in V$, whose masses are the smaller and the larger value, respectively, of the mass of peak i and its complement w.r.t. the parent mass. The relevance $r(v)$ of a vertex v is the relevance of the corresponding peak assigned by the mowers. The reason for generating pairs (x_i, y_i) of vertices is that if a peak is a true peak, then it is either a prefix (a b–ion) or a suffix (a y–ion). Moreover, if the spectrum was perfect, it would also contain its complement.

We use a mass tolerance ε, and if two vertices have the same mass within the mass tolerance ε, then we merge them, and assign the new vertex the maximum relevance value among the merged vertices. The vertices are sorted such that $m(x_0) < m(x_1) < \ldots < m(x_n) < m(y_n) < \ldots < m(y_1) < m(y_0)$[3] . Hereby, x_0 and y_0 are two new vertices with masses $m(x_0) = 1$ and $m(y_0) = \text{parent mass} - 18$, and both relevance 1. At this point, for each pair (x_i, y_i), for $1 \leq i \leq n$, we know that it either constitutes noise, or one is a prefix of the peptide and the other a suffix—but we do not know which is which.

G contains a directed edge (u, v) if $m(v) - m(u)$ can be written as the sum of amino acid masses, within the mass tolerance. We say that a directed path P in G is *k–compatible* if P contains at most one vertex of each pair (x_i, y_i), for $1 \leq i \leq k$. Any n–compatible directed path P in G from x_0 to y_0 corresponds to a solution for the input spectrum, because it represents a partial list of prefixes.

We now show how to fill in a table Q of size $(n+1) \times (n+1)$ that will be used to compute paths from x_0 to y_0. Define $w(P)$, the *pathweight* of a directed path P in G, as $w(P) := \sum_{v \in P} r(v)$. Set

$$\begin{aligned} Q(i,j) := \max\{w(L) + w(R) \mid\ & L \text{ directed path from } x_0 \text{ to } x_i, \\ & R \text{ directed path from } y_j \text{ to } y_0, \\ & \text{and } L \cup R \text{ is } \max(i,j)\text{–compatible}\}. \end{aligned}$$

We set $Q(i, j) = 0$ if no appropriate paths exist. The table Q has the property that $Q(i, j) > 0$ if and only if there is a path L from x_0 to x_i and a path R from y_j to y_0 such that $L \cup R$ is $\max(i, j)$–compatible. The table can be filled in using the crucial observation that the maximum path for a given pair (x_i, y_j), for $i < j$, can be computed using all maximum paths for pairs (x_i, y_k), for $k < j$. Since $j > i$, value y_j can be added to any such pair $L \cup R$ without violating the compatibility condition. The situation is analogous for the case where $i > j$. Thus, $Q(i, j)$ can be computed as follows.

$$Q(i,j) = \begin{cases} \max_{0 \leq k < j}\{Q(i,k) \mid (y_j, y_k) \in E, Q(i,k) > 0\} + r(y_j) & \text{, if } i < j; \\ 0 & \text{, if } i = j; \\ \max_{0 \leq k < i}\{Q(k,j) \mid (x_k, x_i) \in E, Q(k,j) > 0\} + r(x_i) & \text{, if } i > j. \end{cases}$$

[3]Because of the merging of vertices, the new value of n may have decreased. Here, we assume that n denotes the correct number of vertices.

The value of a maximum path is now $r_{\max} = \max\{Q(i,j) \mid (x_i, y_j) \in E\}$. Note that $r_{\max} = 0$ means that there is no feasible solution to the input, i.e., the parent mass cannot be written as a sum of amino acid masses within the given error tolerance ε.

Entry $Q(i,j)$ contains the maximum weight of any compatible path from x_0 to x_i and from y_j to y_0. Thus, the table Q can be used in a backtracking algorithm to recursively enumerate all paths from x_0 to y_0. Moreover, if we use a threshold t we can restrict their enumeration to all paths whose weights are above t. The use of such a threshold allows for pruning the tree of computation generated by the backtracking process in early stages. This makes the time spent in the recursion proportional, not to the total number of possible paths, but to the number of paths that are of interest, i.e., whose weights are above the threshold.

When generating graph G, we have inserted an edge between two vertices if the difference of their masses could be written as the sum of amino acid masses (see above). Hence, we need an efficient way to decide whether a given mass M can be represented by a sum of amino acid masses, and we need to compute all such amino acid sequences. We will study this problem as a combinatorial problem in Chapter 8. Here, it suffices to construct an array of Boolean variables $b_0, \ldots, b_N$ such that variable b_i represents masses $m \in [i \cdot \delta, (i+1) \cdot \delta)$, with δ some small range. Let $m_i = i \cdot \delta + \frac{\delta}{2}$ be the center mass of the interval represented by b_i. The maximum index N depends on the maximum mass $M_{\max}$ considered, and is computed as $N = \lceil \frac{M_{\max}}{\delta} \rceil$. Typically, we use $\delta = 0.01$Da and $M_{\max} = 1000$Da.

The variables b_i are initialized as follows. If the mass interval represented by b_i contains the mass of any single amino acid, then b_i is set to TRUE, otherwise b_i is set to false. This can be done in $20 + N = O(N)$ time, as there are 20 amino acids. In a second phase, we run from b_0 to b_N and set b_i to TRUE if there is an amino acid mass a such that the variable b_j containing mass $m_i - a$ is set to TRUE. The second pass takes $20N = O(N)$ steps.

To answer the question whether a mass M measured with error ε can be represented by a sum amino acid masses, we check all variables b_i that represent part of the interval $(M - \varepsilon, M + \varepsilon)$. If one of them is true, the answer is `YES`, if all are false, then the answer is `NO`. Typically, we use $\varepsilon = 0.5$Da.

Finally, if we want to *enumerate* all amino acid sequences that yield mass M up to error ε, we proceed as follows. For all true b_i's that represent part of the interval $(M - \varepsilon, M + \varepsilon)$, and for all amino acid masses a, we test whether the variable b_j containing $m_i - a$ is true. If so, we store the letter of amino acid a and recursively enumerate all sequences for mass

m_j. This algorithm, however, enumerates all permutations of all possible sequences. To avoid this, in recursion depth d we only consider amino acids whose letters are lexicographically larger or equal to the amino acid letter chosen in depth $d-1$. This way, only distinct sequences with respect to permutation are output.

7.4 Experimental Results

We ran Audens on a test set of 266 MS/MS spectra for which the correct peptide sequence is known. This test set was generated by Grossmann during his diploma thesis [42]. The spectra were created from the proteins Bovine Serum Albumin, Polyribonucleotide nucleotidyltransferase, Cytochrome C and the WD–40 repeat protein MSI1 fused to Glutathione S–transferase. For each of these spectra the software tool Sequest outputs a "good" sequence, i.e., the sequence match is of high quality, indicated by X–$Corr \geq 2.5$ and $DeltaCn \geq 0.1$, and the proposed peptide sequence occurs in the sequence of the corresponding protein in use. This gives strong evidence that the proposed sequences are correct. Details can be found in [42].

We ran Audens for each of these 266 spectra with the same parameter setting. Figure 7.2 shows the values of the most important parameters in our test run. With this parameter setting, Audens outputs the correct sequence, i.e., a matching multi–sequence for 118 out of the 266 given spectra. Moreover, for 79 of these spectra, Audens lists the matching multi–sequence among the first three multi–sequences in its ranked output. For comparison, we ran the de novo sequencing tool Lutefisk [89, 88, 107] on our test spectra as well. Lutefisk output the correct sequence for only 68 of the spectra among its first three candidates. In Chapter A in the Appendix we show for each input spectrum the correct sequence and the best rank for a matching sequence generated by Audens and Lutefisk, respectively. The running time of Audens for a single spectrum was usually below 2 seconds on a common PC with 700MHz and 256 MB RAM.

7.5 Conclusion

The purpose of Audens was to explore the potential of preprocessing MS/MS spectra using different grass mowing heuristics that are derived from chemical properties of amino acids and peptides. To this end, our experimental results show that this approach is very promising. In fact, for approximately 30% of the input spectra the correct sequence was among the first three candidates generated by Audens. However, in its current form Audens

```
Approx2Mono.useIt=yes
ammoniaMower2.massTolerance=0.2
ammoniaMower2.massToleranceComp=0.5
ammoniaMower2.relevanceFactor=20
ammoniaMower2.relevanceFactorComp=30
ammoniaMower2.useIt=yes
complementMower.massTolerance=0.5
complementMower.relevanceFactor=20
complementMower.useIt=no
EgsitMower.EgsitX=40
EgsitMower.relevanceFactor=1
EgsitMower.useIt=yes
intersectionMower.peakTolerance=0.1
intersectionMower.relevanceFactor=20
intersectionMower.useIt=no
isotopeMower.massTolerance=0.1
isotopeMower.numIsotopes=2
isotopeMower.relevanceFactor=30
isotopeMower.useIt=yes
isotopeMower2.massTolerance=0.1
isotopeMower2.massToleranceComp=0.5
isotopeMower2.numIsotopes=1
isotopeMower2.relevanceFactor=20
isotopeMower2.relevanceFactorComp=30
isotopeMower2.useIt=yes
offsetMower.offset=14
offsetMower.parentMassTolerance=0.5
offsetMower.peakTolerance=0.1
offsetMower.relevanceFactor=10.0
offsetMower.useIt=no
PercentageMower.percentage=0.03
PercentageMower.relevanceFactor=50
PercentageMower.useIt=yes
PercentageMower2.massToleranceComp=0.5
PercentageMower2.percentage=0.008
PercentageMower2.relevanceFactor=10
PercentageMower2.relevanceFactorComp=30
PercentageMower2.useIt=yes
sequencer.relativeWeightMin=0.95
thresholdMower.relevanceFactor=20
thresholdMower.threshold=500000.0
thresholdMower.useIt=no
thresholdMower2.massToleranceComp=0.5
thresholdMower2.relevanceFactor=20
thresholdMower2.relevanceFactorComp=20
thresholdMower2.threshold=5000.0
thresholdMower2.useIt=no
waterMower.massTolerance=0.1
waterMower.numIsotopes=1
waterMower.relevanceFactor=40
waterMower.useIt=no
waterMower2.massTolerance=0.2
waterMower2.massToleranceComp=0.5
waterMower2.relevanceFactor=10
waterMower2.relevanceFactorComp=30
waterMower2.useIt=yes
windowMower.massWindow=130.0
windowMower.numPeaks=2
windowMower.relevanceFactor=30.0
windowMower.useIt=yes
windowMower2.massToleranceComp=0.5
windowMower2.massWindow=100.0
windowMower2.numPeaks=4
windowMower2.relevanceFactor=10.0
windowMower2.relevanceFactorComp=30
windowMower2.useIt=yes
```

Figure 7.2: Main parameters of Audens for the test set.

only serves as a “proof of concept”, and there are several ways to improve it:

- The parameter setting we used was found by “trial and error”, i.e., we modified each parameter by hand until we found a good setting. We are sure that more elaborate techniques to find good parameters, like neuronal networks or other machine learning algorithms, will further improve the performance of Audens.

- There are many other mowers that might help in the sequencing process; for instance, an *intersection mower* can be used to virtually merge many spectra that belong to the same peptide. In fact, in our test set there are several spectra for one single peptide, and we can find such spectra using for instance the parent mass or other matching techniques (e.g. as proposed in [87]). Such a merging step might reduce the amount of noise, since – typically – true peaks are more likely to occur in many, or even all, of these spectra, while grass peaks occur in a more randomly distributed way.

- In the current version of Audens, no postprocessing is applied, and the ranking of the multi–sequences in the output is exclusively based on the relevances of the corresponding peaks. Of course, additional ranking schemes can be used to determine the quality of the multi–sequences in the output of Audens (see for instance [53, 72]), thus allowing to exclude "bad" candidates.

- In order to get more reliable sequencing results, additional experiments can be performed, such as methyl ester derivatisation of the peptides, where each y–ion is shifted by 14Da. Similar to the intersection mower sketched above, an *offset mower* can be implemented that makes use of such peak shifts. First experiments have already been performed in [42].

In fact, based on our results an interdisciplinary project with biologists and computer scientists was launched at ETH Zurich that aims at the development of a de novo peptide sequencing tool.

Chapter 8

Mass Decomposition

8.1 Introduction

In this chapter, we study the computational complexity of DECOMPOSITION, where we are given a mass M and we ask whether it can be decomposed into amino acid masses. We recapitulate the definition from the introduction (cf. Definition 1.4.3):

Definition. *Given n positive integers $c_1, \ldots, c_n$ and a positive integer M, are there non–negative integers $\lambda_1, \ldots, \lambda_n$ such that $\sum_{i=1}^{n} \lambda_i \cdot c_i = M$?*

This problem occurs – in a restricted variant – in the de novo sequencing algorithm that we applied in Section 7.3. There we used a simple dynamic programming algorithm to compute all masses M, up to a certain upper bound, that can be decomposed. The question whether a mass can be decomposed into amino acids occurs in the MASS FINDING problem as well: Obviously, we can find mass M as submass in a protein only if it can be decomposed at all. Thus, if we know in advance that M cannot be decomposed, then we do not need to search the databases for M.

Apart from its application in computational biology, the DECOMPOSITION problem is highly motivated from a theoretical point of view, since it is a variation of SUBSET SUM respectively KNAPSACK where we are allowed to use items more than once. This problem is known as INTEGER KNAPSACK problem [40]. The problem has many real world applications, e.g. in electronic cash systems or cargo loading. In fact, DECOMPOSITION is also known as COIN CHANGE problem, where we are given a coin system and a total price, and we ask whether it is possible to pay the price using only these coins (assuming that the amount of coins in the wallet is not limited).

The decision problem Coin Change is NP-complete [57] and can be solved in pseudo–polynomial time using dynamic programming [94]. Apart from that, only little is known about the decision problem. In fact, most literature deals with the minimization variation of the problem, assuming that there is a coin of denomination 1 (this makes the decision problem trivial), and asking for a representation of the price with a minimum number of coins. This minimization problem can again be solved in pseudo–polynomial time using dynamic programming [94]. Other algorithmic approaches use for instance branch and bound techniques or a greedy strategy that always takes the largest coin still possible [60]. There exist coin systems where the greedy algorithm always outputs a minimal solution, e.g. for US coins $1, 5, 10, 25, 50, 100$, but in general, it can be arbitrarily far away from optimum [60]. Deciding whether the greedy algorithm outputs an optimal solution for every price is possible in time polynomial in the number of coins [68], while it is coNP-hard to decide for a specific total x whether the greedy algorithm produces a minimal result [54]. For the k–payment problem, where we have to find a minimal set of coins such that k exact payments, not known in advance, can be performed, lower and upper bounds and an efficient algorithm are known [67]. Recently, optimal coin systems – that allow to minimize the average number of coins needed for a set of payments – have been proposed [80]. For a survey we refer the reader to the book by Martello and Toth that devotes an entire chapter to the Coin Change problem [60].

In the first part of this chapter, we study how the computational complexity of Decomposition is related to the size of the input. Here, the size of the input is measured in the number of amino acid masses n and the logarithm of the total mass M. We first observe in Section 8.2 that Decomposition can be solved in polynomial time if the number of amino acids is constant. In particular, this is the case if we restrict Decomposition to the 20 most common amino acids. However, there exists a huge set of post–translational modifications, such as phosphorylation or glycosylation, that can virtually change the masses of amino acids, which increases the number of different masses to be considered [5, 51]. Hence, it is natural to consider n to be non–constant. Moreover, the "algorithm" that solves Decomposition for constant n has worst case running time highly exponential in n; in fact, it can be more than $2^{(n-1)^2}$, which yields unreasonable running times even for small values of n.

In Section 8.3, we study how the *size* of the amino acid masses in the input affects the complexity of Decomposition. The monoisotopic amino acid masses are known very precisely. However, accuracy of mass spectrometry measurements in typical experimental settings can typically vary between zero and five post decimal digits. We can assume that the amino acid masses are given in the corresponding precision as well. If we multiply all masses by an appropriate power of 10, in order to obtain integer inputs, then values we use for different experiments can vary between 10^2 for low precision experiments, and 10^7 for experiments with high precision. This motivates to distinguish between “small” and “large” numbers in the input of DECOMPOSITION.

We first study the case where most of the input numbers are small, i.e., bounded by a polynomial in n. If the total mass M itself is small, then obviously all relevant amino acid masses are small as well (otherwise, the corresponding amino acids do not fit and can be ignored), and the pseudo–polynomial time algorithm by Wright [94] yields immediately a polynomial time algorithm for such restricted input. In the opposite case, if all amino acid masses are small, then this does not immediately imply that the total mass M has to be small as well. This is different for instance for PARTITION, where small input numbers imply immediately that each partition has small total as well. We give an algorithm that solves DECOMPOSITION in polynomial time for small amino acid masses and arbitrarily large total mass M. For the algorithm, we first observe that if M is not a multiple of the greatest common divisor of the amino acid masses, then it cannot be decomposed. Otherwise, M can be decomposed if it is sufficiently large. Our algorithm uses this fact to either output the result immediately for large masses, or to apply a pseudo–polynomial time subroutine for small masses.

Then we extend this algorithm and show that the DECOMPOSITION problem can even be solved in polynomial time in the presence of few large amino acids, i.e., if there is only a constant number of amino acid masses that are super–polynomial in n.

In the second part of this chapter, we turn to the search problem of DECOMPOSITION, where we are not only asked whether mass M can be decomposed, but to output a solution $\lambda_1, \ldots, \lambda_n$, if any. In the context of database search, such a solution can be helpful in the development of fast protein identification algorithms. If there are many decompositions for one mass, then we can ask for a decomposition with specific requirements. Two very natural requirements are decompositions with a minimum or a maximum number of amino acids, yielding a lower respectively upper bound on the number of amino acids in which the mass can be decomposed. These two optimization problems are referred to as MIN DECOMPOSITION and MAX DECOMPOSITION and defined as follows.

Definition 8.1.1 (MIN DECOMPOSITION)**.** *Given a positive integer M and positive integers $c_1, \ldots, c_n$, find non–negative integers $\lambda_1, \ldots, \lambda_n$ with $\sum_{i=1}^{n} \lambda_i \cdot c_i = M$ such that $\sum_{i=1}^{n} \lambda_i$ is minimum.*

Definition 8.1.2 (MAX DECOMPOSITION). *Given a positive integer M and positive integers $c_1, \ldots, c_n$, find non–negative integers $\lambda_1, \ldots, \lambda_n$ with $\sum_{i=1}^{n} \lambda_i \cdot c_i = M$ such that $\sum_{i=1}^{n} \lambda_i$ is maximum.*

The variation of DECOMPOSITION where we ask for a maximum mass that is less than a given upper bound B and that can be decomposed is a special case of INTEGER KNAPSACK. For this problem pseudo–polynomial time algorithms and approximation algorithms are known [13]. However, we are not aware of any approximability results for MIN DECOMPOSITION, and of no results at all for MAX DECOMPOSITION. For this reason we study the approximability of MIN DECOMPOSITION and MAX DECOMPOSITION in Section 8.4. We present for both problems a gap–producing reduction from PARTITION that shows that no polynomial time algorithm can exist that has any constant approximation ratio, unless $\mathsf{P} = \mathsf{NP}$.

8.2 DECOMPOSITION with Few Amino Acids

For two amino acids, the problem DECOMPOSITION can be solved using the extended version of Euclid's algorithm. The following lemma shows that integer linear programming can be used to solve the problem in polynomial time even if the number of amino acids is any constant.

Lemma 8.2.1. DECOMPOSITION, MIN DECOMPOSITION *and* MAX DECOMPOSITION *can be solved in polynomial time if the number of amino acids is constant.*

Proof: For all three problems there is a straightforward formulation as an integer linear program (ILP). E.g. for MIN DECOMPOSITION, this is

$$\begin{aligned}
\min \sum_{i=1}^{n} \lambda_i, & & \\
\text{subject to } \sum_{i=1}^{n} \lambda_i c_i = M & & \\
\lambda_i \geq 0 & & \text{for } 1 \leq i \leq n \\
\lambda_i \text{ integer} & & \text{for } 1 \leq i \leq n
\end{aligned}$$

The claim follows immediately from the fact that integer linear programs are solvable in polynomial time if the number of variables (in this case, n) in the program is constant [77].

□

Observe that the running time to solve integer linear programs with n variables can be more than $2^{(n-1)^2}$ [77].

8.3 DECOMPOSITION with Small Amino Acids

In this section, we study how the size of the amino acid masses affects the complexity of DECOMPOSITION. We say that an amino acid mass is *large* if it is super–polynomial in the number n of amino acids in the input, and the mass is *small* if it is polynomially bounded in n. We present an algorithm that solves the DECOMPOSITION problem in polynomial time if the number of large amino acid is at most constant. This algorithm combines the pseudo–polynomial algorithm from the book by Martello and Toth [60] with the following result by Brauer [12].

Fact 8.3.1. *[12] Given n positive integers $a_1, \ldots, a_n$ with $a_1 \leq \ldots \leq a_n$ and $\gcd(a_1, \ldots, a_n) = 1$, and a positive integer $f \geq a_1 \cdot a_n$. Then the equation $\sum_{i=1}^{n} \mu_i a_i = f$ has a solution with non–negative integers μ_i, for $1 \leq i \leq n$.*

Observe that $\sum_{i=1}^{n} \mu_i a_i = f$ cannot have an integer solution if f is not a multiple of $g = \gcd(a_1, \ldots, a_n)$, since the left hand side is obviously divisible by g.

We now present our algorithm for the DECOMPOSITION problem. Given amino acid masses $c_1, \ldots, c_n$ and total mass M, let $g = \gcd(c_1, \ldots, c_n)$. We assume w.l.o.g. that $c_1 < \ldots < c_n$. If $g = 1$, then let $b = c_1 \cdot c_n$. If $M > b$, then mass M can be decomposed, due to Fact 8.3.1. If $M \leq b$, then we use the pseudo–polynomial algorithm from the book by Martello and Toth [60] to solve the problem explicitly. In order to make this work self–contained, we present a simplified version of this algorithm here:

> Let F be a Boolean array of length $M + 1$. We set $F(m)$ to TRUE, for $0 \leq m \leq M$, if mass m can be decomposed into amino acids. Then $F(0) =$ TRUE, and $F(m) =$ TRUE if and only if $F(m - c_i)$ is TRUE for at least one $i \in \{1, \ldots, n\}$. Obviously, the total mass M can be decomposed if and only if $F(M)$ is TRUE. This algorithm has pseudo–polynomial running time $O(n \cdot M)$.

Since all c_i are polynomially bounded in n, we have $M \leq b = c_1 \cdot c_n = O(poly(n))$, hence, for this case our algorithm runs in polynomial time.

If $g > 1$, then every mass M that is not a multiple of g cannot be represented. On the other hand, if g is a divisor of M, then $M = \sum_{i=1}^{n} \lambda_i \cdot c_i$

is equivalent to $M' = \sum_{i=1}^{n} \lambda_i \cdot c'_i$, with $c'_i = \frac{c_i}{g}$ and $M' = \frac{M}{g}$. Since $\gcd(c'_1, \ldots, c'_n) = 1$, the last equation can be decided using one recursive call of our algorithm. This yields the following lemma.

Lemma 8.3.2. *The* DECOMPOSITION *problem can be solved in polynomial time if the size of all amino acid masses is polynomially bounded in the number n of amino acid masses.*

We now show how to solve the DECOMPOSITION problem if exactly one amino acid mass is large. Subsequently, we will generalize this to any constant number of large amino acids.

Lemma 8.3.3. *Given $n + 1$ amino acid masses $c_1, \ldots, c_n, h$ with $c_i = O(\textit{poly } (n))$, for $1 \leq i \leq n$, and $h = \omega(\textit{poly } (n))$, and a total mass M. Then the* DECOMPOSITION *problem can be solved in time $O(\textit{poly } (n))$.*

Proof:
Let $g = \gcd(c_1, \ldots, c_n)$ and $b = c_1 \cdot c_n$. Then $b < h$. If $\gcd(c_1, \ldots, c_n, h)$ does not divide M, then the total mass cannot be decomposed. On the other hand, if g divides M, then Fact 8.3.1 applies, and the mass can be decomposed using only the small amino acid masses. Otherwise, we use the following case distinction to prove the claim:

- If $M < h$, then mass h cannot be used in any decomposition of M; hence, Lemma 8.3.2 immediately yields the claim.
- If $M = h$, then the problem is trivial.
- If $M \leq 2gh$, then we can construct $2g$ new instances of DECOMPOSITION as follows. Amino acid masses are $c_1, \ldots, c_n$, and the total mass is $M_k := M - k \cdot h$, for $k \in \{0, \ldots, 2g\}$. Then we have $M = k \cdot h + M_k$, and M can be decomposed if and only if any of these DECOMPOSITION instances yields a decomposition of mass M_k.
- If $M > 2gh$, then $\gcd(g, h)$ divides M, since we have $\gcd(g, h) = \gcd(\gcd(c_1, \ldots, c_n), h) = \gcd(c_1, \ldots, c_n, h)$. Since $M > gh$, there exist non–negative integers α and β such that $M = \alpha \cdot h + \beta \cdot g$ (by Fact 8.3.1). Moreover, by "shifting" multiples of g (replace α by $\alpha - g$ and β by $\beta + h$), we can achieve that $\alpha \leq g$. Then $\beta = \frac{M - \alpha \cdot h}{g} \geq \frac{2gh - gh}{g} = h > b$. Hence, there exist $\lambda_1, \ldots, \lambda_n$ such that $\beta \cdot g = \sum_{i=1}^{n} \lambda_i \cdot c_i$.

$\square$

In the following theorem, we extend the previous algorithm to any number of large amino acid masses. This increases the running time drastically;

however, if the number of large amino acid masses is only constant, then this still yields an algorithm polynomial in n.

Theorem 8.3.4. *Given are $n+k$ amino acid masses $c_1, \ldots, c_n, h_1, \ldots, h_k$ and a total M, such that $c_1 < \ldots c_n$, $c_n = O(poly\ (n))$, and $h_1 < \ldots \leq h_k$, $h_1 = \omega(poly\ (n))$. Then the* DECOMPOSITION *problem can be solved in time $O((c_1+1)^{k-1} poly\ (n))$.*

Proof: If $M < h_j$ for some $j \in \{1, \ldots, k\}$, then mass h_j cannot be used in any decomposition of M; moreover, the problem becomes trivial if $M = h_j$. Hence, we can assume w.l.o.g. that $M > h_k$.

We use an inductive argument over k to prove this lemma. Obviously, Lemma 8.3.3 yields the claim for $k = 1$. For $k \geq 2$, we do the following: If $\gcd(c_1, \ldots, c_n, h_1, \ldots, h_k)$ does not divide M, then the total mass M cannot be decomposed. Hence, we can assume that $\gcd(c_1, \ldots, c_n, h_1, \ldots, h_k)$ divides M. Let $b = c_1 \cdot h_k$.

If $M > b$, then Fact 8.3.1 applies, and M can be decomposed. Otherwise, we have $M \leq b = c_1 \cdot h_k$. Hence, if M can be decomposed, then mass h_k can occur at most c_1 times. Thus, for all $\beta_k \in \{0, \ldots, c_1\}$, we can check recursively whether $M' := M - \beta_k \cdot h_k$ can be decomposed into the amino acid masses $c_1, \ldots, c_n, h_1, \ldots, h_{k-1}$. If so, then this immediately yields a decomposition of M. Otherwise, there is no solution for any of the β_k's, and M cannot be decomposed.

For the running time, let $T(\ell)$ be the time required to solve the DECOMPOSITION problem with only the first ℓ large amino acid masses $h_1, \ldots, h_\ell$. Then $T(1) = O(\text{poly }(n))$, due to Lemma 8.3.3. Moreover, we have the recursive equation $T(k) \leq (c_1+1) \cdot T(k-1)$. This yields $T(k) \leq (c_1 + 1)^{k-1} \cdot T(1)$, which proves the claim.

□

Observe that the previous algorithm achieves running time polynomial in n if the number of large amino acids is constant. Moreover, we can solve the corresponding search problem of DECOMPOSITION as well, since we can modify the pseudo–polynomial algorithm such that it outputs an appropriate solution, if any.

8.4 Inapproximability of MIN DECOMPOSITION and MAX DECOMPOSITION

In this section, we show that the minimization and the maximization variation of DECOMPOSITION cannot be approximated to within any constant factor, unless $\mathsf{P} = \mathsf{NP}$. To establish this result we present gap–producing reductions from ALTERNATING PARTITION (see Definition 2.4.7).

Theorem 8.4.1. MIN DECOMPOSITION *cannot be approximated to within any constant factor, unless* $\mathsf{P} = \mathsf{NP}$.

Proof: Let $(u_1, v_1), \ldots, (u_n, v_n)$ be an instance of ALTERNATING PARTITION. Let $s = \sum_{i=1}^{n}(u_i + v_i)$, and ω be any positive integer. Moreover, we define a total mass $M = \langle \frac{s}{2}, \frac{s}{2} \rangle \circ \mathbb{1}_n \circ \langle \omega \rangle$ and amino acid masses

$$\begin{array}{llll}
a_i = \langle u_i, v_i \rangle & \circ\, \Delta_n(i) & \circ\, \langle 0 \rangle & \text{for } 1 \le i \le n \\
b_i = \langle v_i, u_i \rangle & \circ\, \Delta_n(i) & \circ\, \langle 0 \rangle & \text{for } 1 \le i \le n \\
c = \langle 0, 0 \rangle & \circ\, \mathbb{0}_n & \circ\, \langle \omega \rangle & \\
d = \langle 0, 0 \rangle & \circ\, \mathbb{0}_n & \circ\, \langle 1 \rangle & \\
e = \langle \frac{s}{2}, \frac{s}{2} \rangle & \circ\, \mathbb{1}_n & \circ\, \langle 1 \rangle &
\end{array}$$

For these numbers we use base $Z = 2 \cdot (n+1) \cdot (s + \omega + 1)$ to avoid any carry–overs from one digit to the next in the following additions. The amino acid masses a_i, b_i, c, d and e and the total mass M are an instance of MIN DECOMPOSITION. Observe that $M = e + \omega \cdot c$. We now show the following properties:

1. If there is a solution for the ALTERNATING PARTITION instance, then total mass M can be decomposed with $n+1$ amino acids.

2. If there is no solution for the ALTERNATING PARTITION instance, then total mass M cannot be decomposed with less than ω amino acids.

For the first implication, let I and J be a solution for the ALTERNATING PARTITION instance. Then the amino acid masses a_i with $i \in I$ and the amino acid masses b_j with $j \in J$ together sum up to $\frac{s}{2}$ in the first digit, and the same holds for the second digit. Hence, adding amino acid mass c to these amino acids yields total mass M, using $n+1$ amino acids.

To prove the second implication, assume that there is no solution for the ALTERNATING PARTITION instance. In any solution for the MIN DECOMPOSITION instance, we can use at most one of the amino acids a_i and b_i, since both have a 1 in the $(i+2)$'nd digit, and this digit is set to 1 in total mass M. Since no selection of amino acids a_i and b_i can yield sum $\frac{s}{2}$ (otherwise, this would yield a solution for the ALTERNATING PARTITION instance), we must use amino acid e in any solution for the Min Decomposition instance. This implies immediately that $\omega - 1$ amino acid masses d need to be used to obtain total mass M. Hence, we need at least ω amino acids in total (and this bound is tight).

Our two implications show that no polynomial time algorithm can exist with an approximation ratio of less than $\frac{\omega}{n+1}$, unless $\mathsf{P} = \mathsf{NP}$. Choosing appropriate ω proves the claim.

$\square$

We now give a similar reduction that proves that Max Decomposition is hard to approximate as well.

Theorem 8.4.2. Max Decomposition *cannot be approximated to within any constant factor, unless* $\mathsf{P} = \mathsf{NP}$.

Proof: Let $(u_1, v_1), \ldots, (u_n, v_n)$ be an instance of Alternating Partition. Let $s = \sum_{i=1}^{n}(u_i + v_i)$, and ω be any positive integer. Moreover, we define a total mass $M = \langle n, \omega, \frac{s}{2}, \frac{s}{2}\rangle \circ \mathbb{1}_n$ and amino acid masses

$$\begin{aligned}
a_i &= \langle 1, 0, u_i, v_i\rangle && \circ\, \Delta_n(i) && \text{for } 1 \le i \le n\\
b_i &= \langle 1, 0, v_i, u_i\rangle && \circ\, \Delta_n(i) && \text{for } 1 \le i \le n\\
c &= \langle 0, 1, 0, 0\rangle && \circ\, \mathbb{0}_n\\
d &= \langle n, \omega, \frac{s}{2}, \frac{s}{2}\rangle && \circ\, \mathbb{1}_n
\end{aligned}$$

We use base $Z = 2 \cdot (n+1) \cdot (s + \omega + 1)$ for these numbers to avoid any carry–overs from one digit to the next in the following additions. Observe that $d = M$. The amino acid masses a_i, b_i, c, and d, and the total mass M build an instance of Max Decomposition with the following properties:

1. If there is a solution for the Alternating Partition instance, then total mass M can be decomposed with $n + \omega$ amino acids.
2. If there is no solution for the Alternating Partition instance, then total mass M cannot be decomposed with more than one amino acid.

For the first claim, let I and J be a solution for the Alternating Partition instance. Then the amino acid masses a_i with $i \in I$ and the amino acid masses b_j with $j \in J$ together sum up to $\frac{s}{2}$ in the third digit, and the same holds for the fourth digit. Hence, adding amino acid c to these amino acids ω times yields total mass M, using $n + \omega$ amino acids.

For the second implication, assume that there is no solution for the Alternating Partition instance. In any solution for the Max Decomposition instance, we can use at most n of each of the masses a_i and b_i, since otherwise the first digit becomes greater than n. Since our base in each digit is sufficiently large, we cannot have any carry–overs in the last $n+2$ digits in any solutions. A solution can use at most one of the amino acids a_i and b_i, since both have a 1 in the $(i+4)$'th digit, and this digit is set to 1 in total mass M. Since no selection of amino acids a_i and b_i can yield sum $\frac{s}{2}$ (otherwise this would yield a solution for the Alternating Partition instance), we must use amino acid d in any solution for the Max Decomposition instance; but this yields already the total mass M. Hence, only one amino acid can be used in a solution, namely d.

Thus, no polynomial time algorithm can exist that achieves an approximation ratio of less than $\frac{n+\omega}{1}$, unless $\mathsf{P} = \mathsf{NP}$.

□

8.5 Conclusion

We have shown that Decomposition can be solved in polynomial time if the number of amino acid masses is constant, or if all amino acid masses except a constant number are polynomially bounded in n, the number of amino acids. On the other hand, the optimization variations Min Decomposition and Max Decomposition, where we ask for solutions of minimum respectively maximum cardinality, cannot be approximated by any constant ratio, unless $\mathsf{P} = \mathsf{NP}$, even if M always has at least one decomposition. Our results evoke different directions of future research:

- In how many different ways can a certain mass be decomposed if we use the true amino acid masses? Is there an upper bound on this number if we assume a certain precision?
- Our algorithm for the case of a constant number of amino acid masses is asymptotically rather inefficient; in fact, its worst case running time is highly exponential in the number of amino acids. How does this algorithm perform in practice? Are there more efficient (combinatorial) algorithms for this problem variation?
- Are there other interesting optimization variations of Decomposition (of theoretical or practical impact) that can be at least approximated?

Chapter 9

Summary and Conclusion

9.1 Results

In this thesis, we have given efficient algorithms, NP-hardness proofs, and inapproximability results for several problems that arise in the realm of restriction site experiments; furthermore, we have presented Audens, the prototype of a software tool for de novo peptide sequencing. In the following, we summarize our results and draw final conclusions. Unless stated otherwise, our inapproximability results hold if $\mathsf{P} \neq \mathsf{NP}$.

We first studied how to reconstruct the ordering of fragments of a DNA molecule, given data from double or partial digestion experiments. We introduced variations of the DOUBLE DIGEST problem and the PARTIAL DIGEST problem that model different types of errors, in particular additions, omissions or measurement errors. Our results are summarized in the following two tables, where $|D|$ denotes the cardinality of a distance multiset D, and $\varepsilon > 0$ a constant. To prove these results, we have shown in addition that DISJOINT ORDERING is NP-complete.

Double Digestion		
Problem:	**Hardness Result:**	**Algorithm:**
Double Digest	strongly NP-hard	NP
Disjoint Double Digest	strongly NP-hard	NP
Min Absolute Error Double Digest	no finite approximation ratio	
Min Relative Error Double Digest	no approximation ratio $\frac{877}{876}$	approximation ratio 2
Min Point Number Double Digest	no approximation ratio $\frac{392}{391}$	approximation ratio 3
any optimization variation of Disjoint Double Digest	no finite approximation ratio	

Partial Digestion		
Problem:	**Hardness Result:**	**Algorithm:**
Min Partial Digest Superset	NP-hard	
t–Partial Digest Superset	NP-hard for any $t = \Omega(\|D\|^{\frac{1}{2}+\varepsilon})$	
Max Partial Digest Subset	no approximation ratio $\|D\|^{\frac{1}{2}-\varepsilon}$, unless NP = ZPP	approximation ratio $O(\|D\|^{\frac{1}{2}})$
Partial Digest With Errors	strongly NP-hard	NP

To prove NP-hardness of Min Partial Digest Superset, we used a reduction from Equal Sum Subsets. Motivated by this link between the two problems, we introduced and studied several natural variations of Equal Sum Subsets. Our results are shown in the following table, where S denotes the sum of all elements in the input set A.

Equal Sum Subsets		
Problem:	**Hardness Result:**	**Algorithm:**
FACTOR–r SUM SUBSETS	NP-hard for any rational $r > 0$	NP
k EQUAL SUM SUBSETS	NP-hard for any $k \geq 2$	$O(\frac{nS^k}{k^{k-1}})$
k EQUAL SUM SUBSETS	strongly NP-hard for $k = \Omega(n)$	NP
kESS OF CARDINALITY c		$O(n^{kc})$
kESS OF SPECIFIED CARDINALITY	NP-hard	$O(\frac{S^k \cdot n^{k+1}}{k^{2k-1}})$
kESS OF EQUAL CARDINALITY	NP-hard	$O(\frac{S^k \cdot n^{k+1}}{k^{2k-1}})$
ESS OF DIFFERENT CARDINALITY	NP-hard	NP
ESS WITH EXCLUSIONS	NP-hard	$O(n^2 \cdot S)$
ESS WITH ENFORCED ELEMENT	NP-hard	NP
ALTERNATING EQUAL SUM SUBSETS	NP-hard	NP
ESS FROM TWO SETS	NP-hard	NP
ESS OF EQUAL CARDINALITY FROM TWO SETS	NP-hard	NP
ESS WITH DISJOINT INDICES FROM TWO SETS	NP-hard	NP
ESS WITH DISJOINT COVERING INDICES FROM TWO SETS	NP-hard	NP
ESS WITH IDENTICAL INDICES FROM TWO SETS	NP-hard	NP

In the second part of this thesis, we addressed the problem of protein identification by digestion experiments. First we studied the problem to find a given submass in a protein, which is the MASS FINDING problem. This problem arises naturally when proteins are searched in large protein databases using their mass fingerprint. There are two simple algorithms that solve the MASS FINDING problem: LINSEARCH, which has linear query time and does not need any additional data structure; and HASHING, which has constant query time, but needs an additional hash table that can require quadratic storage space. We presented algorithm LOOKUP, which beats these two simple algorithms in a sense, as it has sublinear query time $O(\frac{n}{\log n})$, and it needs only linear storage space. However, LOOKUP has no practical impact, as it requires unreasonably huge inputs to be efficient.

A different approach to identifying a protein is to establish its amino acid sequence using MS/MS data. We have implemented the prototype of a de novo peptide sequencing tool, called Audens. This tool is based on a sequencing algorithm from the literature and uses several heuristics (grass mowers) to distinguish between noise peaks and true peaks in a spectrum. We applied Audens to a set of 266 test spectra, showing that the prototype can already sequence approximately one third of the spectra.

Finally, we studied the DECOMPOSITION problem, where we want to decompose a given mass into amino acid masses. Our results are shown in the following table, where c_1 denotes the minimum amino acid mass in the input.

Decomposition		
Problem:	**Hardness Result:**	**Algorithm:**
DECOMPOSITION		polynomial for constant number of amino acid masses
DECOMPOSITION		$O((c_1+1)^{k-1} \cdot \text{poly}(n))$ for n small and k large amino acid masses
MIN DECOMPOSITION	no constant approximation ratio	
MAX DECOMPOSITION	no constant approximation ratio	

9.2 Open Problems

We have shown results for a large set of combinatorial problems that are more or less connected to restriction site experiments. Of course, several questions are still open, and we have presented and discussed them already in the corresponding chapters of this thesis. However, there are three major challenges for future research that we would like to re–emphasize here:

PARTIAL DIGEST**:** The PARTIAL DIGEST problem has been studied extensively in the literature, and a huge amount of knowledge has been accumulated about the problem: For instance, several variations of the problem have been shown to be NP-hard; different algorithmic approaches have been proposed to solve the problem in practice, e.g. backtracking algorithms and pseudo–polynomial algorithms; and, finally, the number of possible solutions for one instance has been characterized. However, the status of the PARTIAL DIGEST problem itself is still open, and it is an intriguing challenge – at least from the point of view of theoretical computer science – to determine the exact computational complexity of this combinatorial problem.

MASS FINDING**:** We have shown that it is possible to solve the MASS FINDING problem in sublinear time with only linear additional storage space. However, the running time of our algorithm LOOKUP is only just sublinear, and moreover, it is only of theoretical impact. Therefore, the design of more efficient algorithms that allow for fast search in weighted strings is an important open problem.

DE NOVO SEQUENCING**:** De novo protein sequencing plays an important role in proteomics, and fast and reliable software tools are required to allow for protein identification in high–throughput experiments. Our prototype Audens is a first step in this direction, but new techniques – for instance integrating additional experimental data – will be required to develop tools that perform well in practice.

Biology easily has 500 years of exciting problems to work on...[1]

...and computational biology as well!

[1]Excerpt from an interview with Donald E. Knuth at Computer Literacy Bookshops, December 1993.

Appendix A

Audens Results

On the following pages, we list the experimental results for our software tool Audens. The test set consisted of 266 spectra. For each spectrum three files were present: A `.dta`–file, which contains the spectrum itself in ASCII–format; a `.out`–file, which contains the output of Sequest for the spectrum; and a `.lut`–file, which contains the output of Lutefisk for the spectrum. We briefly describe the contents of these files. The first line of a `.dta`–file contains the parent mass and the charge state of the peptide; each of the following lines denotes a mass/charge ratio, and the corresponding abundance, sorted in ascending order according to the mass/charge ratio. A `.out`–file contains some general information about the spectrum, and a ranked list of sequences that match the given spectrum best, together with the quality of the match. Finally, a `.lut`–file constains the sequences generated by Lutefisk, together with their rank.

The meaning of the columns in the following table is as follows. All data can be found on the accompanying CD–ROM.

No. An identification number for the spectrum.

Parent Mass The total peptide mass from the `.out`–file.

Number of Peaks The total number of peaks in the spectrum.

Sum of Abundances The total sum of the abundances of all peaks in the spectrum.

Maximum Abundance The maximum value among the abundances of all peaks in the spectrum.

Correct Sequence The correct amino acid sequence that corresponds to the spectrum. This sequence was identified by Sequest as “good” match, i.e., the sequence match is of high quality, indicated by *X–Corr* ≥ 2.5 and *DeltaCn* ≥ 0.1, and the sequence occurs as a substring in the sequence of the corresponding protein in use.

Lutefisk Rank The first position of a sequence in the output of Lutefisk that matches the correct sequence for the spectrum. If Lutefisk did not find the correct sequence at all, no rank is given.

AudensRank The first position of a multi–sequence in the ranked output of Audens that matches the correct sequence for the spectrum. If Audens did not find the correct sequence in the list at all, no rank is given.

No.	Parent Mass	Number of Peaks	Sum of Abundances	Maximum Abundance	Correct Sequence	Lutefisk Rank	Audens Rank
1	1101.27	232	5218981	452344	AAVAGIAMGLVK	1	1
2	1101.34	400	116381336	11182139	AAVAGIAMGLVK		1
3	1100.76	330	31849716	3236658	AAVAGIAMGLVK	1	1
4	1101.12	138	1582988	161611	AAVAGIAMGLVK		1
5	1101.28	392	95618921	9689095	AAVAGIAMGLVK		1
6	1101.2	423	57441580	5370974	AAVAGIAMGLVK		1
7	1101.12	113	1177712	116762	AAVAGIAMGLVK	1	2
8	1101.05	295	10265323	977754	AAVAGIAMGLVK	1	1
9	1100.86	252	3818844	370254	AAVAGIAMGLVK	1	1
10	1100.81	185	27436912	2739349	AAVAGIAMGLVK	1	1
11	1100.8	198	2142945	175818	AAVAGIAMGLVK	2	1
12	922.95	116	2410042	665566	AEFVEVTK		
13	922.82	106	1660662	459456	AEFVEVTK	1	
14	922.97	90	1255463	354393	AEFVEVTK		
15	1567.87	102	393439	33130	DAFLGSFLYEYSR		
16	1568.41	299	17398640	930821	DAFLGSFLYEYSR		5
17	1569.43	201	2155410	177943	DAFLGSFLYEYSR		
18	1568.41	256	75363669	3925059	DAFLGSFLYEYSR		2
19	1569.61	133	414618	19959	DAFLGSFLYEYSR		
20	1568.31	263	12158801	782331	DAFLGSFLYEYSR		2
21	1375.33	225	4555188	248035	DAQVLDELMGER		
22	1376.03	113	476398	32041	DAQVLDELMGER		
23	1376.22	234	3400757	159992	DAQVLDELMGER		
24	1376.02	175	881363	55542	DAQVLDELMGER		
25	1376.29	170	928070	62009	DAQVLDELMGER		2
26	1377.75	101	342260	15052	DAQVLDELMGER		
27	1049.31	237	2133213	204717	DDISQFAPR		
28	1191.07	87	421860	68150	DGISALQMDIK		7
29	1191.05	103	433177	51239	DGISALQMDIK		
30	1190.9	159	1168605	108110	DGISALQMDIK		1
31	1190.72	45	167538	37791	DGISALQMDIK	1	
32	1190.93	137	802029	108534	DGISALQMDIK		1
33	1191.08	106	535853	85434	DGISALQMDIK		1
34	1191.74	183	4070280	679252	DGISALQMDIK	1	3
35	974.87	203	2840557	626960	DLGEQHFK		
36	1392	199	2717962	196798	EALTLPSGDFVSR		
37	1392.25	276	4118741	368829	EALTLPSGDFVSR		
38	1310.48	182	2124058	340036	EGLVHISQIADK		1
39	1310.16	342	11154321	1609644	EGLVHISQIADK		
40	1310.3	163	1487161	265165	EGLVHISQIADK		1
41	1310.28	257	7516873	1092320	EGLVHISQIADK		1

No.	Parent Mass	Number of Peaks	Sum of Abundances	Maximum Abundance	Correct Sequence	Lutefisk Rank	Audens Rank
42	1310.37	333	12066865	1727450	EGLVHISQIADK		1
43	1310.1	324	3090101	378343	EGLVHISQIADK		20
44	1310.14	190	4920833	882079	EGLVHISQIADK	3	10
45	1311.68	218	9856783	867008	EGLVHISQIADK		
46	1415.78	375	2665238	401498	EGRPSEGETLIAR		
47	1415.47	223	1047195	120582	EGRPSEGETLIAR		
48	1415.39	296	2015660	268176	EGRPSEGETLIAR		
49	1415.22	217	1172252	231295	EGRPSEGETLIAR		
50	1414.95	299	1850264	295020	EGRPSEGETLIAR		
51	1415.17	269	1317707	118498	EGRPSEGETLIAR		7
52	1415.3	267	1593750	143468	EGRPSEGETLIAR		
53	1245.35	197	4782192	483616	EIMQVALNQAK		
54	1245.13	170	1347993	164598	EIMQVALNQAK		
55	1245.17	381	25944615	3288799	EIMQVALNQAK		2
56	1244.63	127	471778	70977	EIMQVALNQAK		
57	1245.02	417	17716711	1311489	EIMQVALNQAK	1	
58	1245.08	147	2067667	233545	EIMQVALNQAK		
59	1245.99	175	1317072	155398	EIMQVALNQAK		
60	1245.22	331	41144933	4832087	EIMQVALNQAK		
61	1244.74	74	351299	44528	EIMQVALNQAK		35
62	1762.21	573	5229322	448375	FEENSTNSTKSFKIK		
63	1250.42	163	6787076	929365	FKDLGEEHFK		15
64	1250.05	318	3850754	566350	FKDLGEEHFK		14
65	1250.14	131	964721	151490	FKDLGEEHFK		
66	1249.73	119	877671	166966	FKDLGEEHFK	1	
67	1249.88	331	4510344	546886	FKDLGEEHFK		
68	1250.1	221	1961746	302269	FKDLGEEHFK		7
69	1249.9	146	1027924	145179	FKDLGEEHFK	1	
70	1250.08	321	5467588	510490	FKDLGEEHFK		6
71	1250.19	333	8726260	1034869	FKDLGEEHFK		2
72	1250.55	142	655749	101065	FKDLGEEHFK		
73	1747.79	574	4029991	408970	FLQLAPGEYFFSSIK		
74	991.77	264	6987434	682194	GDISEFAPR	1	2
75	991.89	352	45959373	3815442	GDISEFAPR		2
76	991.58	139	1816390	186254	GDISEFAPR	1	2
77	992.36	312	91759677	9010372	GDISEFAPR	1	2
78	991.66	200	10310723	1051485	GDISEFAPR	2	10
79	992.04	247	3822624	363911	GDISEFAPR		4
80	991.91	311	13347488	1312446	GDISEFAPR		2
81	991.95	234	3999622	411684	GDISEFAPR		2
82	991.67	399	60442462	6395552	GDISEFAPR		2
83	992.03	270	7777338	796881	GDISEFAPR	3	2
84	1489.44	242	3373604	170038	GETQALVTATLGTAR		2
85	1489.16	283	3108750	152708	GETQALVTATLGTAR	4	
86	1489.26	196	778436	29598	GETQALVTATLGTAR		83
87	1488.69	303	6838972	259429	GETQALVTATLGTAR		
88	1488.73	175	777213	58446	GETQALVTATLGTAR		
89	1489.23	346	10056838	409759	GETQALVTATLGTAR	3	21
90	1489.08	234	2114110	122715	GETQALVTATLGTAR		
91	1489.07	267	1936668	116878	GETQALVTATLGTAR		53
92	1488.65	129	308147	9250	GETQALVTATLGTAR		
93	1490.13	614	14677280	718180	GETQALVTATLGTAR		
94	1490.17	223	1022684	43262	GETQALVTATLGTAR		
95	1434.56	161	1150440	142924	HKTGPNLHGLFGR		
96	1306.17	362	17629203	3049876	HLVDEPQNLIK		1
97	1305.93	306	4771654	790244	HLVDEPQNLIK		
98	1305.89	409	25391080	4222395	HLVDEPQNLIK	1	
99	1306.22	203	78654551	13862228	HLVDEPQNLIK		23
100	1306.49	276	5708277	691444	HLVDEPQNLIK		
101	1305.84	380	28835127	4313670	HLVDEPQNLIK	1	64
102	1305.86	208	2074957	274188	HLVDEPQNLIK	3	
103	1307.12	371	8611892	1316697	HLVDEPQNLIK		
104	1306.27	220	44641888	7829228	HLVDEPQNLIK		12
105	1306.18	347	31067179	3884365	HLVDEPQNLIK		1
106	1306.09	178	1927853	326582	HLVDEPQNLIK	1	2
107	1306.28	382	13410342	2104557	HLVDEPQNLIK		1
108	1306.27	216	34279244	5681378	HLVDEPQNLIK		4
109	1306.37	259	4656845	766234	HLVDEPQNLIK		
110	1889.45	255	15903802	2888065	HPYFYAPELLYYANK		

No.	Parent Mass	Number of Peaks	Sum of Abundances	Maximum Abundance	Correct Sequence	Lutefisk Rank	Audens Rank
111	1889.51	296	14832815	2885945	HPYFYAPELLYYANK		
112	1889.6	361	21823278	4302996	HPYFYAPELLYYANK		
113	1889.5	240	1828273	530771	HPYFYAPELLYYANK		
114	1890.36	205	1524748	307518	HPYFYAPELLYYANK		
115	1599.43	168	804386	91023	IATDPFVGNLTFFR		
116	1359.44	146	965012	88319	IEEITAEIEVGR		2
117	1340.07	219	2321082	482566	INPDKIKDVIGK		
118	997.36	104	803539	54886	IPALDLLIK		
119	1351.1	93	329136	26570	IVDFGAFVAIGGGK		1
120	1623.3	279	5030841	633837	KLTVDKSMVEVFVK		
121	1143.18	269	2413725	207263	KQTALVELLK		2
122	1143.31	359	10656373	1123932	KQTALVELLK	4	
123	1143.1	174	1381387	152748	KQTALVELLK	2	1
124	1142.93	344	7036967	615640	KQTALVELLK	1	
125	1143.17	133	719341	68736	KQTALVELLK		4
126	1143.35	232	1342492	178529	KQTALVELLK		1
127	1143.59	220	12677984	1635413	KQTALVELLK		3
128	1143.58	390	37039011	3220219	KQTALVELLK		1
129	1143.2	260	54549029	5162768	KQTALVELLK		
130	1143.44	201	2157879	245619	KQTALVELLK	1	
131	1841.57	170	631278	32689	KTGQAPGFTYTDANKNK		
132	1640.62	253	2208355	386672	KVPQVSTPTLVEVSR		
133	1640.37	265	9888964	1724486	KVPQVSTPTLVEVSR		
134	1640.27	195	4206954	734925	KVPQVSTPTLVEVSR		
135	1640.11	193	6814348	1383960	KVPQVSTPTLVEVSR	4	
136	1641.79	230	6971738	1050883	KVPQVSTPTLVEVSR		
137	1640.51	312	24532260	2590299	KVPQVSTPTLVEVSR		
138	1174.3	94	547283	83464	LAAITAQDSQR		8
139	1480.4	471	5946140	428190	LGEYGFQNAILVR	1	15
140	1480.21	113	396180	33003	LGEYGFQNAILVR		
141	1480.25	588	20440960	1413676	LGEYGFQNALIVR	1	23
142	1480.44	493	13353268	1062111	LGEYGFQNALIVR	1	1
143	1480.07	349	32116824	2515777	LGEYGFQNALIVR	1	
144	1480.39	323	27906188	1946269	LGEYGFQNALIVR		43
145	1480.53	354	143865113	12010592	LGEYGFQNALIVR		1
146	1480.23	354	241183707	18412240	LGEYGFQNALIVR	2	25
147	1480.17	350	42924707	3087914	LGEYGFQNALIVR		1
148	1480.5	326	17637399	1411367	LGEYGFQNALIVR	1	62
149	1662.56	171	1357449	58575	LHILGVMEQAINAPR	1	1
150	1662.1	134	521563	32101	LHILGVMEQAINAPR		67
151	1592.15	300	3823170	286711	LKGADPEDVIMGAFK		
152	1163.6	367	15490257	3280194	LVNELTEFAK	1	
153	1164.26	137	1437112	286863	LVNELTEFAK	1	
154	1164.31	338	16218718	3366894	LVNELTEFAK	2	
155	1164.19	369	88364995	19709252	LVNELTEFAK	2	21
156	1163.87	169	2957471	688006	LVNELTEFAK	1	7
157	1164.21	366	20262851	2767056	LVNELTEFAK	3	2
158	1164.24	273	3466539	569443	LVNELTEFAK	1	
159	1164.14	318	5426201	1185362	LVNELTEFAK	1	
160	1164	333	57973152	12509409	LVNELTEFAK		11
161	1164.09	363	34588743	7397680	LVNELTEFAK		2
162	1163.97	129	715583	123448	LVNELTEFAK	2	
163	1163.63	285	8380758	1785486	LVNELTEFAK	1	13
164	1163.85	379	31951593	6675687	LVNELTEFAK	1	4
165	1164.7	294	3306539	781814	LVNELTEFAK		
166	1164.32	309	7436405	1515348	LVNELTEFAK	2	
167	1164.04	231	298761025	61313968	LVNELTEFAK		5
168	1164.07	363	589600759	58965824	LVNELTEFAK		2
169	1165.09	287	337613313	64245984	LVNELTEFAK		
170	1164.7	348	194537398	35617720	LVNELTEFAK		54
171	1164.09	275	3186357	669366	LVNELTEFAK	1	
172	1164.04	206	154447355	30479544	LVNELTEFAK		2
173	1164.53	216	26200329	5318261	LVNELTEFAK	2	
174	1497.36	134	457164	40581	MPTLEEYGTNLTK		
175	1393.23	264	4142735	305934	NAKVLFVGTTGVGK		
176	1530.58	380	4201150	290018	NHLDSQKLTAFKK		
177	1530.57	134	789805	95272	NHLDSQKLTAFKK		
178	1812.1	277	3331231	642913	PGQDFFPLTVNYQER		
179	1812.15	262	5450817	996666	PGQDFFPLTVNYQER		

No.	Parent Mass	Number of Peaks	Sum of Abundances	Maximum Abundance	Correct Sequence	Lutefisk Rank	Audens Rank
180	1811.45	214	1612547	347954	PGQDFFPLTVNYQER		
181	1072.9	304	2653820	286593	PSEGETLIAR		1
182	1072.99	227	1471248	143060	PSEGETLIAR		1
183	1072.84	283	2532992	235373	PSEGETLIAR	1	1
184	1073.12	242	1521036	169063	PSEGETLIAR	2	1
185	1073.18	236	1472454	130014	PSEGETLIAR		1
186	1700.76	145	512737	35158	QGVVVITGASSGLGLAAAK		
187	1440.34	251	1843309	147349	RHPEYAVSVLLR		75
188	1440.48	258	1419467	113842	RHPEYAVSVLLR		
189	1440.7	267	85834729	6159524	RHPEYAVSVLLR		13
190	1440.7	259	25334150	2263296	RHPEYAVSVLLR		1
191	1440.85	313	4228131	282631	RHPEYAVSVLLR		
192	1440.51	259	4084212	338041	RHPEYAVSVLLR		1
193	1748.4	416	2111911	213452	RIFTYNNEFKVTSK		
194	1465.77	364	7623659	417421	SCGLAIGTTIVDADK		
195	1050.32	171	8469265	2435044	SLDAQQIFK		1
196	1471.64	262	35698148	9928458	TGQAPGFTYTDANK		
197	1471.73	121	730469	166964	TGQAPGFTYTDANK		
198	1713.7	352	27118288	3299589	TGQAPGFTYTDANKNK		
199	1714.46	269	1575378	128546	TGQAPGFTYTDANKNK		
200	989.84	246	13338725	2898579	THGSAIFTR		
201	989.87	221	10387982	1603527	THGSAIFTR		
202	989.88	166	2342955	390251	THGSALFTR		
203	989.62	267	23330635	4564001	THGSALFTR		
204	1400.03	326	4582261	622377	TVMENFVAFVDK		
205	1400.85	146	818877	120019	TVMENFVAFVDK		
206	1400.12	138	897432	115188	TVMENFVAFVDK		
207	1400.32	230	80018330	6951927	TVMENFVAFVDK	1	9
208	1400.06	288	70579859	5896518	TVMENFVAFVDK	1	
209	1400.36	377	188904493	18778152	TVMENFVAFVDK		
210	1400.26	204	3795096	313049	TVMENFVAFVDK		
211	800.55	163	9789601	1605642	VAALAEAR	1	2
212	800.78	162	117092483	19746788	VAALAEAR		3
213	800.87	91	1870575	315594	VAALAEAR		2
214	800.61	167	83072235	13838918	VAALAEAR	1	1
215	800.95	162	68495616	11323035	VAALAEAR		2
216	800.65	159	5848833	955757	VAALAEAR	1	3
217	800.55	145	4783640	744761	VAALAEAR	1	1
218	800.6	159	43469517	7353214	VAALAEAR	1	1
219	800.94	165	7975553	1323211	VAALAEAR		3
220	1789.69	191	615285	33458	VISWYDNEWGYSNR		
221	1514.97	192	6456923	670465	VPEVSTPTLVEVSR		
222	1512.1	151	912309	96613	VPQVSTPTLVEVSR		
223	1512.34	243	2856964	329296	VPQVSTPTLVEVSR		
224	1512.54	134	702555	90208	VPQVSTPTLVEVSR		
225	1512.21	178	1396933	150472	VPQVSTPTLVEVSR		
226	1512.02	137	878806	115158	VPQVSTPTLVEVSR		
227	1512.27	220	1481686	182176	VPQVSTPTLVEVSR		
228	1512.61	156	687650	79163	VPQVSTPTLVEVSR		
229	1512.47	217	14337669	1614714	VPQVSTPTLVEVSR		
230	1512.43	237	17080750	1769950	VPQVSTPTLVEVSR		
231	1608.12	141	461177	39141	VTDYLQMGQEVPVK		
232	1607.43	225	2417243	237609	VTDYLQMGQEVPVK	3	
233	1608.36	350	35006374	2653659	VTDYLQMGQEVPVK		
234	1607.04	367	8439404	1076102	VTDYLQMGQEVPVK		
235	1606.97	414	46306879	4405254	VTDYLQMGQEVPVK	2	
236	1607.27	249	6695163	589292	VTDYLQMGQEVPVK		
237	1607.25	312	2422989	197924	VTDYLQMGQEVPVK		19
238	1607.01	122	456669	39320	VTDYLQMGQEVPVK		
239	1607.14	439	24072050	2197103	VTDYLQMGQEVPVK	4	
240	1607.09	73	352563	53343	VTDYLQMGQEVPVK	2	6
241	1608.71	284	12482428	1073945	VTDYLQMGQEVPVK		
242	1738.09	310	2873065	187161	VYSGVVNSGDTVLNSVK		
243	1739.28	205	635047	34754	VYSGVVNSGDTVLNSVK		
244	1739.27	213	939440	75588	VYSGVVNSGDTVLNSVK		
245	1738.18	275	1619262	117619	VYSGVVNSGDTVLNSVK		
246	1825.34	486	18041888	4178698	WDWQPEPVNEALNAR		
247	1824.92	505	55161315	13979193	WDWQPEPVNEALNAR		
248	1825.14	380	3847509	797632	WDWQPEPVNEALNAR		

No.	Parent Mass	Number of Peaks	Sum of Abundances	Maximum Abundance	Correct Sequence	Lutefisk Rank	Audens Rank
249	1825.98	301	2277924	372668	WDWQPEPVNEALNAR		
250	1826.24	190	8954519	2139014	WDWQPEPVNEALNAR		
251	935.83	212	5048149	665883	YAQVDVIK	1	1
252	936.12	204	5741630	1285244	YAQVDVIK	1	3
253	936.05	198	4954870	1016352	YAQVDVIK		2
254	936.45	162	1860739	364323	YAQVDVIK		
255	935.93	185	3475274	740387	YAQVDVIK		1
256	936.07	208	11512769	2622576	YAQVDVIK		1
257	936.16	200	4064426	876050	YAQVDVIK	1	
258	936.28	137	1410174	248807	YAQVDVIK		
259	936.08	196	2708518	568313	YAQVDVIK	1	1
260	935.65	226	6551632	1377314	YAQVDVIK	2	3
261	936.33	154	1990424	346642	YAQVDVIK	1	
262	1331	187	803046	51756	YGGGANTLAAGYSK		2
263	928	112	3377608	606025	YLYEIAR		2
264	928.24	127	33910241	6877158	YLYEIAR	1	2
265	927.79	120	3593599	968364	YLYEIAR	1	
266	1460.12	628	26566627	1964782	YSEIYYPTVPVK		

Appendix B

CD Contents

File	**Description**
`AuDeNS`	The de novo sequencing tool Audens, implemented in Java. To start Audens, change to directory AuDeNS, adapt (if necessary) the paths in `config.txt` and `StartAudens.bat`, and run `StartAudens.bat`. The file `config.txt` includes the parameter setting used in the thesis for the test data set.
`DATA`	The 266 spectra from the test set. For each spectrum, there are three important files: `.dta`, the spectrum itself; `.out`, the Sequest output; `.lut`, the Lutefisk output. In addition, there is a `.msq` file, which contains temporary information from Audens, and a `.info` file, which was generated by our statistics tool. The file `AuDeNSOutput.txt` contains the results generated by Audens. For each spectrum, the highest ranked multi-sequence that matches the correct sequence is given. *Remark*: There is no `.lut` file for spectrum PNP_coverage2_50fmol.0835.0838.2.dta, as Lutefisk generated a *divide by zero* error for this spectrum.
`VNGYSEIER.dta`	The example spectrum used in the thesis.
`Results.xls`	A summary of the results of Sequest, Lutefiskand Audens for each spectrum. The ordering is identical to the table in Appendix A of the thesis.
`Thesis`	The text of the thesis, in various file formats.
`Readme.txt`	This text.

Appendix C

Problem Definition Index

Problem	Definition	Page
3–Partition	2.4.1	25
Alternating Equal Sum Subsets	5.1.8	70
Alternating Partition	2.4.7	26
Decomposition	1.4.3	19
Disjoint Double Digest	3.1.1	28
Disjoint Ordering	1.4.1	14
Double Digest	1.2.1	4
Equal Sum Subsets	1.4.2	16
ESS From Two Sets	5.1.10	71
ESS Of Different Cardinality	5.1.9	71
ESS Of Equal Cardinality From Two Sets	5.1.11	71
ESS With Disjoint Covering Indices From Two Sets	5.1.11	71
ESS With Disjoint Indices From Two Sets	5.1.11	71
ESS With Enforced Element	5.1.7	70
ESS With Exclusions	5.1.6	70
ESS With Identical Indices From Two Sets	5.1.11	71
Exact 3–Satisfiability	2.4.4	25
Factor–r Sum Subsets	5.1.1	68
k Equal Sum Subsets	5.1.2	68
kESS Of Cardinality c	5.1.3	69
kESS Of Equal Cardinality	5.1.5	69
kESS Of Specified Cardinality	5.1.4	69
Mass Finding	1.3.1	8

Bibliography

[1] F. Alizadeh, R. M. Karp, L. A. Newberg, and D. K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. In *Proc. of the* 4th *SIAM–ACM Symposium on Discrete Algorithms (SODA 1993)*, pages 371–381, 1993.

[2] L. Allison and C. N. Yee. Restriction site mapping is in separation theory. *Computer Applications in the Biosciences*, 4(1):97–101, 1988.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, editors. *Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

[4] V. Bafna and N. Edwards. SCOPE: A probabilistic model for scoring tandem mass spectra against a peptide database. *Bioinformatics*, 17(Supplement 1):13–21, 2001.

[5] S. Baginsky. Personal communication, 2003. ETH Zurich, Institute of Plant Sciences.

[6] S. Baginsky, M. Cieliebak, W. Gruissem, T. Kleffmann, Zs. Lipták, M. Müller, and P. Penna. AuDeNS: A tool for automatic de novo peptide sequencing. Technical Report 383, ETH Zurich, Department of Computer Science, 2002.

[7] C. Bazgan, M. Santha, and Zs. Tuza. Efficient approximation algorithms for the subset–sum equality problem. In *Proc. of the* 25th *International Colloquium on Automata, Languages and Programming (ICALP 1998)*, pages 387–396, 1998.

[8] B. Bellon. Construction of restriction maps. *Computer Applications in the Biosciences*, 4(1):111–115, 1988.

[9] P. J. Bentley. *Programming Pearls.* Addison–Wesley, 1986.

[10] J. Błażewicz, P. Formanowicz, M. Kasprzak, M. Jaroszewski, and W. T. Markiewicz. Construction of DNA restriction maps based on a simplified experiment. *Bioinformatics*, 17(5):398–404, 2001.

[11] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS–PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31, 2003.

[12] A. Brauer. On a problem of partitions. *American Journal of Mathematics*, 64:299–312, 1942.

[13] A. K Chandra, D. S. Hirschberg, and C. K. Wong. Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, 3:293–304, 1976.

[14] T. Chen, M.-Y. Kao, M. Tepel, J. Rush, and G. M. Church. A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. In *Proc. of the* 11^{th} *SIAM–ACM Symposium on Discrete Algorithms (SODA 2000)*, pages 389–398, 2000.

[15] J. Chlebíková. Personal communication, 2002. University of Kiel, Theory of Parallelism.

[16] J. Chlebíková and M. Chlebík. Inapproximability results for bounded variants of optimization problems. In *Proceedings of the* 14^{th} *International Symposium on Fundamentals of Computation Theory (FCT 2003)*, pages 27–38, 2003.

[17] J. Chlebíková and M. Chlebík. Inapproximability results for bounded variants of optimization problems. Technical Report TR03–026, Electronic Colloquium on Computational Complexity, 2003.

[18] M. Cieliebak, S. Eidenbenz, and A. Pagourtzis. Composing equipotent teams. In *Proc. of the* 14^{th} *International Symposium on Fundamentals of Computation Theory (FCT 2003)*, pages 98–108, 2003.

[19] M. Cieliebak, S. Eidenbenz, A. Pagourtzis, and K. Schlude. Equal sum subsets: Complexity of variations. Technical Report 370, ETH Zurich, Department of Computer Science, 2002.

[20] M. Cieliebak, S. Eidenbenz, and P. Penna. Noisy data make the partial digest problem NP-hard. Technical Report 381, ETH Zurich, Department of Computer Science, 2002.

[21] M. Cieliebak, S. Eidenbenz, and P. Penna. Noisy data make the partial digest problem NP-hard. In *Proc. of the* 3^{rd} *Workshop on Algorithms in Bioinformatics (WABI 2003)*, pages 111–123, 2003.

[22] M. Cieliebak, S. Eidenbenz, and G. J. Woeginger. Double digest revisited: Complexity and approximability in the presence of noisy data. Technical Report 382, ETH Zurich, Department of Computer Science, 2002.

[23] M. Cieliebak, S. Eidenbenz, and G. J. Woeginger. Double digest revisited: Complexity and approximability in the presence of noisy data. In *Proc. of the* 9^{th} *International Computing and Combinatorics Conference (COCOON 2003)*, pages 519–527, 2003.

[24] M. Cieliebak, T. Erlebach, Zs. Lipták, J. Stoye, and E. Welzl. Algorithmic complexity of protein identification: Combinatorics of weighted strings. Accepted for special issue of Discrete Applied Mathematics (DAM) devoted to COSSAC 2001, to appear 2003.

[25] M. Cieliebak, T. Erlebach, Zs. Lipták, J. Stoye, and E. Welzl. Algorithmic complexity of protein identification: Combinatorics of weighted strings. Technical Report 361, ETH Zurich, Department of Computer Science, 2001.

[26] M. Cieliebak, T. Erlebach, Zs. Lipták, J. Stoye, and E. Welzl. Algorithmic complexity of protein identification: Searching in weighted strings. In *Proc. of the* 2^{nd} *IFIP International Conference on Theoretical Computer Science (TCS 2002)*, pages 143–156, 2002.

[27] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[28] T Dakić. *On the Turnpike Problem.* PhD thesis, Simon Fraser University, 2000.

[29] V. Dančík, T. A. Addona, K. R. Clauser, J. E. Vath, and P. A. Pevzner. De novo peptide sequencing via tandem mass spectrometry: A graph–theoretical approach. In *Proc. of the* 3^{rd} *Annual International Conference on Computational Molecular Biology (RECOMB 1999)*, pages 135–144, 1999.

[30] K. J. Danna and D. Nathans. Specific cleavage of simian virus 40 DNA by restriction endonuclease of hemophilus influenzal. *Proc. of the National Academy of Sciences of the USA*, 68:2913–2917, 1971.

[31] T. I. Dix and D. H. Kieronska. Errors between sites in restriction site mapping. *Computer Applications in the Biosciences*, 4(1):117–123, 1988.

[32] D-Z. Du and F. K. Hwang, editors. *Combinatorial Group Testing and its Applications.* World Scientific, second edition, 2000.

[33] J. Eng, A. McCormack, and J. R. Yates III. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry (JASMS).*, 5:976–989, 1994.

[34] D. Fasulo. *Algorithms for DNA Restriction Mapping.* PhD thesis, University of Washington, 2000.

[35] D. Fenyo, J. Qin, and B. T. Chait. Protein identification using mass spectrometric information. *Electrophoresis*, 19:998–1005, 1998.

[36] J. Fernandez-de Cossio, J. Gonzalez, T. Takao, Y. Shimonishi, G. Padron, and V. Besada. A software program for the rapid sequence analysis of unknown peptides involving modifications, based on MS/MS data. In *Proc. of the* 45^{th} *ASMS Conference on Mass Spectrometry and Allied Topics*, 1997.

[37] H.I. Field, D. Fenyo, and R.C. Beavis. RADARS, a bioinformatics solution that automates proteome mass spectral analysis, optimises protein identification, and archives data in a relational database. *Proteomics*, 2(1):36–47, 2002.

[38] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[39] J. Fütterer. Personal communication, 2002. ETH Zurich, Institute of Plant Sciences.

[40] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of* NP*-Completeness.* Freeman, 1979.

[41] L. Goldstein and M. S. Waterman. Mapping DNA by stochastic relaxation. *Advances in Applied Mathematics*, 8:194–207, 1987.

[42] J. Grossmann. Protein identification using mass spectrometry: Development of an approach for automated de novo sequencing. Master's thesis, ETH Zurich, Department of Biology, 2003.

[43] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[44] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[45] W. J. Henzel, T. M. Billeci, J. T. Stults, S. C. Wong, C. Grimley, and C. Watanabe. Identifying proteins from two–dimensional gels by molecular mass searching of peptide fragments in protein sequence databases. *Proc. of the National Academy of Sciences of the USA*, 90(11):5011–5015, 1993.

[46] D. Hochbaum, editor. *Approximation Algorithms for* NP*-Hard Problems.* PWS Publishing Company, 1996.

[47] J. Hromkovic. *Algorithmics for Hard Computing Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics.* Springer, 2001.

[48] J. Inglehart and P. C. Nelson. On the limitations of automated restriction mapping. *Computer Applications in the Biosciences*, 10(3):249–261, 1994.

[49] P. James, editor. *Proteome Research: Mass Spectrometry.* Springer, 2001.

[50] P. James, M. Quadroni, E. Carafoli, and G. Gonnet. Protein identification by mass profile fingerprinting. *Biochemical and Biophysical Research Communications*, 195(1):58–64, 1993.

[51] C. Kannicht, editor. *Posttranslational Modifications of Proteins.* Humana Press, 2002.

[52] M.-Y. Kao, J. Samet, and W.-K. Sung. The enhanced double digest problem for DNA physical mapping. In *Proc. of the* 7th *Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, pages 383–392, 2000.

[53] A. Keller, A. I. Nesvizhskii, E. Kolker, and R. Aebersold. Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search. *Analytical Chemistry*, 74:5383–5393, 2002.

[54] D. Kozen and S. Zaks. Optimal bounds for the change–making problem. *Theoretical Computer Science*, 123(2):377–388, 1994.

[55] P. Lemke, S. S. Skiena, and W. Smith. Reconstructing sets from interpoint distances. Technical Report TR2002–37, DIMACS, 2002.

[56] P. Lemke and M. Werman. On the complexity of inverting the autocorrelation function of a finite integer sequence, and the problem of locating n points on a line, given the $\binom{n}{2}$ unlabelled distances between them. Preprint 453, Institute for Mathematics and its Application IMA, 1988.

[57] G. S. Lueker. Two NP-complete problems in nonnegative integer programming. Technical Report TR–178, Computer Science Laboratory, Princeton University, 1975.

[58] M. Mann, R. C. Hendrickson, and A. Pandey. Analysis of proteins and proteomics by mass spectrometry. *Annual Reviews Biochemistry*, 70:437–473, 2001.

[59] M. Mann and M. Wilm. Error–tolerant identification of peptides in sequence databases by peptide sequence tags. *Analytical Chemistry*, 66(24):4390–4399, 1994.

[60] S. Martello and P. Toth. *Knapsack Problems.* John Wiley & Sons, 1990.

[61] D. R. Martin. Equivalence classes for the double–digest problem with coincident cut sites. *Journal of Computational Biology*, 1(3):241–253, 1994.

[62] E. W. Mayr, H. J. Prömel, and A. Steger. *Lectures on Proof Verification and Approximation Algorithms.* Springer, 1998.

[63] L. Newberg and D. Naor. A lower bound on the number of solutions to the probed partial digest problem. *Advances in Applied Mathematics*, 14:172–183, 1993.

[64] G. Pandurangan and H. Ramesh. The restriction mapping problem revisited. *Journal of Computer and System Sciences*, 65(3):526–544, 2002. Special issue on Computational Biology.

[65] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48:498–532, 1994.

[66] D. J. C. Pappin, P. Højrup, and A. J. Bleasby. Rapid identification of proteins by peptide–mass fingerprinting. *Current Biology*, 3(6):327–332, 1993.

[67] B. Patt-Shamir, Y. Tsiounis, and Y. Frankel. Exact analysis of exact change: The k–payment problem. *SIAM Journal on Discrete Mathematics*, 13(4):436–453, 2000.

[68] D. Pearson. A polynomial–time algorithm for the change–making problem. Technical Report TR 94–1443, Cornell University, 1994.

[69] D. N. Perkins, D. J. C. Pappin, D. M. Creasy, and J. S. Cottrell. Probability based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20:3551–3567, 1999.

[70] P. A. Pevzner. DNA physical mapping and alternating Eulerian cycles in colored graphs. *Algorithmica*, 13:77–105, 1995.

[71] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach.* MIT Press, 2000.

[72] P. A. Pevzner, V. Dančík, and C. L. Tang. Mutation–tolerant protein identification by mass spectrometry. *Journal of Computational Biology*, 7(6):777–787, 2000.

[73] P. A. Pevzner, Z. Mulyukov, V. Dančík, and C. L. Tang. Efficiency of database search for identification of mutated and modified proteins via mass spectrometry. *Genome Research*, 11(2):290–299, 2001.

[74] P. A. Pevzner and M. S. Waterman. Open combinatorial problems in computational molecular biology. In *Proc. of the* 3^{rd} *Israel Symposium on Theory of Computing and Systems (ISTCS 1995)*, pages 158–173, 1995.

[75] J. Rosenblatt and P. Seymour. The structure of homometric sets. *SIAM Journal of Algorithms and Discrete Mathematics*, 3(3):343–350, 1982.

[76] W. Schmitt and M. S. Waterman. Multiple solutions of DNA restriction mapping problems. *Advances in Applied Mathematics*, 12:412–427, 1991.

[77] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1986.

[78] D. B. Searls. Formal grammars for intermolecular structure. In *Proc. of the* 1st *International Symposium on Intelligence in Neural and Biological Systems (INBS'95)*, pages 30–37, 1995.

[79] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Boston, 1997.

[80] J. Shallit. What this country needs is an 18¢ piece. *Mathematical Intelligencer*, 25(2):20–23, 2003.

[81] S. S. Skiena, W. Smith, and P. Lemke. Reconstructing sets from interpoint distances. In *Proc. of the* 6th *ACM Symposium on Computational Geometry (SoCG 1990)*, pages 332–339, 1990.

[82] S. S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56:275–294, 1994.

[83] H. O. Smith and K. W. Wilcox. A restriction enzyme from hemophilus influenza. I. purification and general properties. *Journal of Molecular Biology*, 51:379–391, 1970.

[84] A. P. Snyder. *Interpreting Protein Mass Spectra – A Comprehensive Resource.* Oxford University Press, 2000.

[85] G. Srinivasan, C. M. James, and J. A. Krzycki. Pyrrolysine encoded by UAG in archaea: charging of a UAG–decoding specialized tRNA. *Science*, 296:1459–1462, 2002.

[86] W. Staudenmann and P. James. Interpreting peptide tandem mass–spectrometry fragmentation spectra. In P. James, editor, *Proteome Research: Mass Spectrometry*, pages 143–165. Springer, 2001.

[87] D. L. Tabb, M. J. MacCoss, C. C. Wu, S. D. Anderson, and J. R. Yates III. Similarity among tandem mass spectra from proteomic experiments: Detection, significance, and utility. *Analytical Chemistry*, 75:2470–2477, 2003.

[88] J. A. Taylor and R. S. Johnson. Sequence database searches via de novo peptide sequencing by tandem mass spectrometry. *Rapid Communications in Mass Spectromtry*, 11:1067–1075, 1997.

[89] J. A. Taylor and R. S. Johnson. Implementation and uses of automated de novo peptide sequencing by tandem mass spectrometry. *Analytical Chemistry*, 73:2594–2604, 2001.

[90] P. Tuffery, P. Dessen, C. Mugnier, and S. Hazout. Restriction map construction using a 'complete sentence compatibility' algorithm. *Computer Applications in the Biosciences*, 4(1):103–110, 1988.

[91] M. S. Waterman. *Introduction to Computational Biology*. Chapman & Hall, 1995.

[92] Ingo Wegener. *Komplexitätstheorie*. Springer, 2003.

[93] G. J. Woeginger and Z. L. Yu. On the equal–subset–sum problem. *Information Processing Letters*, 42:299–302, 1992.

[94] J. W. Wright. The change–making problem. *Journal of the ACM*, 22(1):125–128, 1975.

[95] L. W. Wright, J. B. Lichter, J. Reinitz, M. A. Shifman, K. K. Kidd, and P. L. Miller. Computer–assisted restriction mapping: an integrated approach to handling experimental uncertainty. *Computer Applications in the Biosciences*, 10(4):435–442, 1994.

[96] C. H. Wu, H. Huang, L. Arminski, J. Castro-Alvear, Y Chen, Z.-Z. Hu, R. S. Ledley, K. C. Lewis, H.-W. Mewes, B. C. Orcutt, B. E. Suzek, A. Tsugita, C. R. Vinayaka, L.-S. L. Yeh, J. Zhang, and W. C. Barker. The protein information resource: an integrated public resource of functional annotation of proteins. *Nucleic Acids Research*, 30:35–37, 2002.

[97] J. R. Yates III. Database searching using mass spectrometry data. *Electrophoresis*, 19(6):893–900, 1998.

[98] J. R. Yates III, J. K. Eng, and A. L. McCormack. Mining genomes: Correlating tandem mass–spectra of modified and unmodified peptides to sequences in nucleotide databases. *Analytical Chemistry*, 67(18):3202–3210, 1995.

[99] J. R. Yates III, S. Speicher, P. R. Griffin, and T. Hunkapillar. Peptide mass maps: A highly informative approach to protein identification. *Analytical Biochemistry*, 214:397–408, 1993.

[100] Z. Zhang. An exponential example for a partial digest mapping algorithm. *Journal of Computational Biology*, 1(3):235–239, 1994.

[101] `http://us.expasy.org/tools/findmod/findmod_masses.html`.

[102] `http://i-mass.com/guide/aamass.html`.

[103] http://www.appliedbiosystems.com.

[104] http://www.daltonics.bruker.com.

[105] http://www.thermo.com.

[106] http://www.nada.kth.se/∼viggo/wwwcompendium.

[107] http://www.immunex.com/researcher/lutefisk.

[108] http://www.matrixscience.com.

[109] http://www-hto.usc.edu/∼tingchen/resume.html.

[110] http://pir.georgetown.edu.

[111] http://www.promega.com/guides/re_guide/toc.htm.

[112] http://fields.scripps.edu/sequest.

[113] http://www.expasy.ch/sprot.

Curriculum Vitae

Mark Cieliebak

date of birth:	February 28, 1970
place of birth:	Hagen, Germany
citizenship:	German

Education:

1999 – 2003:	PhD student at Institute of Theoretical Computer Science ETH Zurich, Switzerland advisor: Prof. Dr. Peter Widmayer academic title: Dr. sc. tech.
1990 – 1999:	studies at University of Dortmund, Germany major: Computer Science minor: Mathematics academic title: Diplom–Informatiker (German M. Sc.)
1980 – 1989:	high school in Hagen, Germany degree: Abitur (German high school diploma)
1976 – 1980:	primary school in Hagen, Germany

Publications

- **Noisy Data Make the Partial Digest Problem NP-hard**, with Stephan Eidenbenz and Paolo Penna.
  In *Proceedings of the* 3rd *Workshop on Algorithms in Bioinformatics (WABI 2003).* Springer, LNBI 2812, pp. 111–123, 2003.

- **Composing Equipotent Teams**, with Stephan Eidenbenz and Aris Pagourtzis.
  In *Proceedings of the* 14th *International Symposium on Fundamentals of Computation Theory (FCT 2003).* Springer, LNCS 2751, pp. 98–108, 2003.

- **Solving the Robots Gathering Problem**, with Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro.
  In *Proceedings of the* 30th *International Colloquium on Automata, Languages and Programming (ICALP 2003).* Springer, LNCS 2719, pp. 1181–1196, 2003.

- **Double Digest Revisited: Complexity and Approximability in the Presence of Noisy Data**, with Stephan Eidenbenz and Gerhard J. Woeginger.
  In *Proceedings of the* 9th *International Computing and Combinatorics Conference (COCOON 2003).* Springer, LNCS 2697, pp. 519–527, 2003.

- **Algorithmic Complexity of Protein Identification: Combinatorics of Weighted Strings**, with Thomas Erlebach, Zsuzsanna Lipták, Jens Stoye, and Emo Welzl.
  Accepted for special issue of Discrete Applied Mathematics (DAM) devoted to COSSAC 2001, to appear 2003.

- **Statistical Foundations of De Novo Sequencing**, with Sacha Baginsky, Jonas Grossmann, Wilhelm Gruissem, Torsten Kleffmann,

and Lukas K. Mathis.
In *Proceedings of the Swiss Proteomics Society – 2002 Congress Applied Proteomics.* Fontis Media, pp. 121 – 124, 2002.

- **Algorithmic Complexity of Protein Identification: Searching in Weighted Strings**, with Thomas Erlebach, Zsuzsanna Lipták, Jens Stoye, and Emo Welzl.
  In *Proceedings of the* 2nd *IFIP International Conference on Theoretical Computer Science (TCS 2002).* Kluwer Adacemic Publishers, pp. 143–156, 2002.

- **Gathering Autonomous Mobile Robots**, with Giuseppe Prencipe.
  In *Proceedings of the* 9th *International Colloquium on Structural Information and Communication Complexity (SIROCCO 2002).* Proceedings in Informatics 13, Carleton Scientific, pp. 57 – 72, 2002.

Technical Reports

- **Scheduling with Release Times and Deadlines on a Minimum Number of Machines**, with Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer.
  Technical Report 419, ETH Zurich, Department of Computer Science, August 2003.

- **The Weber Point can be Found in Linear Time for Points in Biangular Configuration**, with Luzi Anderegg and Giuseppe Prencipe.
  Technical Report TR–03–01, Universitá di Pisa, Dipartimento di Informatica, January 2003.

- **Double Digest Revisited: Complexity and Approximability in the Presence of Noisy Data**, with Stephan Eidenbenz and Gerhard J. Woeginger.
  Technical Report 382, ETH Zurich, Department of Computer Science, November 2002.

- **AuDeNS: A Tool for Automatic De Novo Peptide Sequencing**, with Sacha Baginsky, Wilhelm Gruissem, Torsten Kleffmann, Zsuzsanna Lipták, Matthias Müller, and Paolo Penna.
  Technical Report 383, ETH Zurich, Department of Computer Science, October 2002.

- **Noisy Data Make the Partial Digest Problem NP-hard**, with Stephan Eidenbenz and Paolo Penna.
  Technical Report 381, ETH Zurich, Department of Computer Science, October 2002.

- **Gathering Autonomous Mobile Robots in Non Totally Symmetric Configurations**, with Giuseppe Prencipe. Technical Report 379, ETH Zurich, Department of Computer Science, October 2002.

- **Equal Sum Subsets: Complexity of Variations**, with Stephan Eidenbenz, Aris Pagourtzis, and Konrad Schlude.
  Technical Report 370, ETH Zurich, Department of Computer Science, July 2002.

- **Algorithmic Complexity of Protein Identification: Combinatorics of Weighted Strings**, with Thomas Erlebach, Zsuzsanna Lipták, Jens Stoye, and Emo Welzl.
  Technical Report 361, ETH Zurich, Department of Computer Science, August 2001.