CHALMERS

A Combinatorial Approach to Protein Mixture Identification based on Mass Spectrometry Data

Master of Science Thesis

MOHSEN NOSRATINIA

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2009

A Combinatorial Approach to Protein Mixture Identification based on Mass Spectrometry Data

Mohsen Nosratinia

Examiner: Peter Damaschke

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2009

Abstract

Protein mixture identification by Mass Spectrometry (MS) data using mass-mapping experiments has become a powerful method in identification, and in some cases quantification, of proteins in samples taken from tissues. The main-stream approaches for database search method usually impose an upper bound of 2 or 3 on the number of proteins in a sample mixture. In this thesis the problem in its most general from is reformulated as a hitting set problem over (finding transversals of) a hypergraph. The limitations and extent of applicability of this approach, considering different error sources in MS data are addressed.

Contents

List of Abbreviations and Symbols

BST	Bounded Search Tree
MALDI	Matrix-Assisted Laser Desorption/Ionization
MS	Mass Spectrometry
PDB	Protein Data Bank
STL	Standard Template Library
TOF	Time Of Flight
LMSE	Least Mean Squared Error
ML	Maximum Likelihood
PDF	Probability Density Function
CDF	Cumulative Density Function
SSE	Sum of Squared Error
f	Maximum number of allowed missing masses
g	Maximum number of allowed spurious masses
K	Maximum cardinality imposed on solutions
N_{iv}	Number of vertices chosen by uniqueness check
N_{uh}	Number of unsettled hyperedges remain after uniqueness check
N_{cp}	Number of candidate proteins
T_{cp}	Time required for creating list of candidate proteins, CP
T_{he}	Time required for creating list of hyperedges, HE
T_b	Branching time

Chapter 1

Introduction

This thesis deals with the identification of proteins in a mixture of peptides. A combinatorial approach is employed with emphasis on transversal theory. The thesis will elaborate to what extend a transversal approach is applicable in this problem.

1.1 Mass Spectrometry

Mass spectrometry (MS) is a widely used analytical technique utilised to measure the mass-to-charge ratio of ions in a sample. The mass-to-charge ratio is then used to generate an spectrum representing the masses of components in the sample.

The general idea is to ionize the molecules of the sample and pass them through a magnetic and/or electric field. The force applied by the field affects the trajectory of the particles. The force is proportional to the charge but the deviation of trajectory is inversely proportional to the mass so the ultimate deviation is directly related to mass-to-charge ratio. Hence, mass spectrometry consists of three major steps and therefore a *Mass Spectrometer* consists of three main blocks to accomplish these steps as follows:

1. A sample of mixture is ionised, usually by loss of an electron. (The *Ion Source* or *Ioniser*)
2. The ions are separated, in space or time, according to their mass and charge (mass-to-charge ratio). (The *Mass Analyser* or *Ion Analyser*)
3. The sorted ions are then detected and the results are reported to data gathering system. (The *Detector*)

The collected data is then analysed by different methods. The introduction of sample and data analysis are sometimes also considered as part of the Mass Spectrometer.

1.1.1 Sample introduction

The technique used for introduction of a specific sample to the ionisation source mostly depends on the ionisation method employed in MS, and also the nature and complexity of the sample.

The sample can either be inserted directly into the ionisation source, or can undergo some sort of chromatography prior to introduction to the ionisation source. The latter method often requires the mass spectrometer to be coupled directly to a high pressure liquid chromatography (HPLC), gas chromatography (GC) or capillary electrophoresis (CE) separation column. In this approach the sample is separated into groups of components. Later on, these components are sequentially entered to the mass spectrometer for individual analysis.

1.1.2 Ionisation methods

The ionisation refers to whole process of adding or removing one or several electron from a particle to create an ion. Ionisation methods used in Mass Spectrometry include the following:

1. Atmospheric Pressure Chemical Ionisation (APCI)
2. Chemical Ionisation (CI)
3. Electron Impact (EI)
4. Electrospray Ionisation (ESI)
5. Fast Atom Bombardment (FAB)
6. Field Desorption / Field Ionisation (FD/FI)
7. Matrix Assisted Laser Desorption Ionisation (MALDI)
8. Thermospray Ionisation (TSP)

Most ionisations methods facilitate creating both positively and negatively charged sample ions. This depends on the proton affinity of the sample.

The most common ionisation methods employed in biochemical analyses are **Electrospray Ionisation (ESI)** and **Matrix Assisted Laser Desorption Ionisation (MALDI)**[**?**]. Ions generated by MALDI are singly charged while ESI-generated ions usually carry multiple charges[**?**].

1.1.3 Mass analyser and ion detector

There are many types of mass analysers depending on the characteristics of the field and the structure of the path ions travel through. The field can be a dynamic or static electric and/or magnetic one. The most common types are[**?**, Chap. 4]:

1. Magnetic-sector MS
2. Time-of-flight (TOF) MS
3. Quadrupole MS
4. Ion-Trap MS
5. Fourier transform MS

The common Mass analyser in mass-mapping applications is TOF.

1.2 Protein Mass Spectrometry

Study of proteins has always been a quintessential part of biological studies. Among different methods used to identify and quantify proteins in biological samples, Mass Spectrometry has long been established as a main method for high-throughput runs. The application of MS in this sense is commonly referred to as *Protein Mass Spectrometry*. A special form of Protein Mass Spectrometry is *peptide mass-mapping* where the protein (or a mixture of proteins) is digested by protease and the MS data reports the molecular weights of the peptides that are produced. This data can be used in several ways to determine the initial protein(s).

1.3 Mathematical preliminaries

1.3.1 Hitting set problem and transversals

The hitting set problem deals with finding a set that contains a representative member from a collection of subsets of a universe M. That means it contains (*hits*) at least one member from each subset. Additionally, the set is required to have no more than K members. In a formal sense:

Definition Let $C = \{c_1, c_2, \ldots, c_n\}$ be a collection of subsets of M, and $0 < K < |M|$ be an arbitrary integer, the problem is to find **hitting sets** $U \subset M$ where $|U| \leq K$ and $U \cap c_i \neq \emptyset$ for all $1 \leq i \leq n$.

A hitting set is **minimal** if none of its proper subsets is also a hitting set. The union of all minimal hitting sets up to a given size is called **full kernel**[**?**].

1.3.2 Hypergraphs

A **hypergraph** is a pair $G = (V, E)$ where V is a set of *vertices* or *nodes* and E is a collection of non-empty subsets of V, called *hyperedges*. That means E is a subset of power set of V and every member of E connects an arbitrary number of vertices in hypergraph.

1.4 The problem of protein mixture identification

Proteins are mostly large molecules that makes them difficult to deal with by most mass spectrometers. This is because of technical difficulties that arise during ionisation of large molecules (commonly larger than 10kDa). Also in analysing a mixture of proteins, interaction of proteins can hinder sample introduction. Therefore, it's a common practice to digest the proteins into small fragments and analyse the fragments. The resulting spectrum is a superposition of spectra of fragments produced by individual proteins. A reference database is prepared *in silico* in advance to facilitate detection of possible proteins in the original mixture. In this thesis trypsin digestion is considered in preparation of reference database.

In the model employed in this thesis proteins are considered as vertices of a hypergraph. Each fragment mass can be produced by a certain set of proteins. This set is a hyperedge over in our hypergraph. Now, given this hypergraph the problem is to find all minimal hitting sets (or transversals) in this hypergraph. So the terms hyperedge and vertex will interchangeably be used for peptide (fragment) mass and protein, respectively.

To achieve this a search tree is built where branches represent the vertices we choose to add to the final solution. We use a bounded search tree because we limit the depth of tree by K.

1.4.1 A small example

Consider a set of proteins with fingerprints corresponding to proteins 1 to 6 in following table and mixtures of proteins with following set of fragmnet masses:

Protein	fragment masses					
ID	35	82	91	133	152	189
1	+	-	-	-	+	-
2	-	+	-	+	-	-
3	-	-	+	-	+	-
4	+	-	-	-	-	+
5	-	-	-	+	-	+
6	+	-	+	-	+	-
Mixture 1	+	-	+	-	+	-
Mixture 2	-	+	+	+	+	-
Mixture 3	+	+	-	+	-	+
Mixture 4	-	-	+	+	+	-
Mixture 4a	*+*	-	+	+	+	-
Mixture 4b	-	*+*	+	+	+	-
Mixture 4c	-	-	+	+	+	*+*
7	+	-	-	+	-	-
Mixture 5	+	+	-	+	-	+

Mixture 1: In this case the mixture matches the spectrum of protein 6, but it also can be produced by digestion of a mixture of proteins 1 and 3. So there are two minimal solutions: $\{6\}$ and $\{1, 3\}$ and full kernel is $\{1, 3, 6\}$.

Mixture 2: Fragment mass 82 only appears in spectrum produced by protein 2. This requires that mass 133 also appears in the mixture, which is consistent. The only other protein that can cover the remaining masses is protein 3. Therefore, there is only one single solution $\{2, 3\}$. This unique correspondence of fragment masses and proteins is employed to make the search space smaller prior to creation of search tree.

Mixture 3: The minimal solution for this case is $\{2, 4\}$. Note that $\{2, 4, 5\}$ is also a solution but is not minimal. This point should be considered in case of generating all possible solutions to include the proteins (vertices) that can be overshadowed in presence of two or more proteins (vertices). Proteins in minimal solution together can generate (hit) all fragment masses (hyperedges) from some other proteins.

Mixture 4: There is no set of proteins that can generate this spectrum. This case can be an instance of missing or spurious masses. In case of spurious masses, the only possibility is that mass 133 doesn't belong to the mixture so the real mixture actually consists of protein 3. In case of missing masses, if we limit the number of missing masses to one, we come up with 3 alternative spectra.

Mixture 4a: Existence of mass 133 requires that either protein 2 or protein 5 be present in the mixture. But protein 2 (5) also generates fragment mass 82 (189) which is not present in this mixture. Therefore, this mixture is inconsistent with current protein database.

Mixture 4b: This mixture is identical to mixture 2 and leads to solution $\{2, 3\}$.

Mixture 4c: Presence of mass 189 requires that either protein 4 or protein 5 be present in the mixture. Protein 4 is inconsistent because of fragment mass 35 so protein 5 is a part of solution. The only protein that can complement protein 5 to create Mixture 4c is protein 3. The only solution is $\{3, 5\}$

Mixture 5, adding protein 7: If protein 7 is also added to the set of proteins and mixture 5 is to be analysed, in case of no errors $\{2, 4\}$ and $\{2, 5, 7\}$ are the answers. Now, if we allow one spurious mass there will be four additional solutions:

- Mass 35 is spurious: $\{2, 5\}$ is a solutions
- Mass 82 is spurious: $\{4, 5\}$ and $\{5, 7\}$ are solutions
- Mass 189 is spurious: $\{2, 7\}$ is a solutions

Chapter 2

Background and methods

2.1 Reference databases

The main fragment mass database is created[1] *in silico* from cleaned SwissProt entries. Each line in the file contains a header that is the protein ID and a series of numbers that correspond to fragment masses from trypsin digestion of the protein. Some fragment masses may appear more than once hence the list of fragment masses are sorted and multiple occurrences are ignored. Note that this may lead to identical sets of masses for two originally distinct proteins[2]. To facilitate cross-reference between masses and proteins, an auxiliary database, *masses database*, is also produced where each entry contains a fragment mass as header followed by a list of ID's of proteins that may produce a given fragment mass.

2.2 Errors in fragment masses

Two most typical kind of errors that arise in using mass spectra are missing masses and spurious masses. The former is generally less likely than the latter one. That is due to the possibility of existence of impurities in original sample or unsuccessful digestions that leads to longer fragments and unwanted masses in final spectrum.

2.2.1 Missing masses

There is a possibility of one or more masses not being observed in MS due to several reasons for instance when digestion was not successful in some sites resulting in larger fragments. For instance a protein that should result in fragments 37 12 49 24 91 can result in masses 37 61 24 91 if the digestion on second trypsin site was not successful.

[1]The original file is provided by Ferdinando Cicalese from Institut für Bioinformatik, Centrum für Biotechnologie (CeBiTec), Universität Bielefeld, Bielefeld, Germany.

[2]In this thesis, protein database refers to this reduced form unless reference to original database is explicitly indicated.

2.2.2 Spurious masses

In some cases there are extra masses observed in MS results mostly because of contamination of the sample, existence of unwanted protein with close mass because of comigration in SDS-GELs and in rare cases partial digestion of some trypsin sites.

2.3 Branching strategies

The underlying idea of finding transversals in the hypergraph generated based on the protein database and fingerprints from mixtures is implemented in three different ways:

- exhaustive-search approach
- hyperedge-oriented approach
- vertex-oriented approach

The **exhaustive-search approach** simply creates one branch for every node in union of all unsettled hyperedges and removes all hyperedges that are settled by adding that node. This makes a cumbersome tree and is only used for comparison. The results of this approach are only a benchmark for the performance of the other approaches.

In **hyperedge-oriented approach**, on every node one unsettled node is chosen and all the vertices contained are selected at once and a new branch is made. This method is based on the construction of a Bounded Search Tree proposed in [**?**, Theorem 6].

Alternatively, a **vertex-oriented approach** may also be used where a vertex is chosen, all hyperedges hit by that vertex are settled and algorithm branches on all vertices in union of remaining unsettled hyperedges. In this approach it's beneficial to start with a vertex that hits many hyperedges.

2.3.1 The algorithm: outline

We used a hyperedge-oriented algorithm. The algorithm can be outlined as follows

```
1. create list of hyperedges $H$
2. create a tree and set the smallest hyperedge in $H$ as root[3]
3. for any node in tree do following steps
   3.1 mark all hyperedges hit by last selected node as settled
   3.2 IF there is no unsettled hyperedge
       3.2.1 add the list of selected vertices up to that node to the list of
             solutions
```

[3]in implementation of the algorithm the hyperedges are sorted by their size and the smallest one is chosen as root

```
    3.3 ELSEIF the number of selected vertices up to that node is $\leq K$
        3.3.1 choose smallest unsettled hyperedge $h$ and mark it settled
        3.3.2 branch on every vertex contained in $h$ by adding it to the list of
              selected vertices
    3.4 ELSE mark as dead end
```

This algorithm works fine as long as there is no error in the sample fragment masses. The algorithm should be revised in case of missing masses errors and spurious masses errors.

2.3.2 The algorithm: missing masses

For the case of missing masses a list of extra hyperedges is also created in parallel. First, a candidate list, say CP, of all proteins that have at most f fragments not present in sample fragment spectrum are created. List of extra hyperedges EH is the set of all hyperedges corresponding to fragment masses associated with proteins in CP not including the masses in the sample. In other words, CP contains all proteins that their spectrum is a subset of sample fragment masses. Let $M(p)$ denote spectrum of protein p, $P(m)$ denote all proteins that contain fragment mass m, P all proteins in database, S set of fragment masses in sample and ES set of extra masses that can be a candidate as a missing mass. Then

$$\begin{aligned} CP &= \{p \in P \mid |M(p) - S| \leq f\} \\ ES &= \bigcup_{p \in CP} M(p) \setminus S \end{aligned}$$

And, initial hyperedges and extra hyperedges are

$$\begin{aligned} H &= \{P(m) \cap CP | m \in S\} \\ EH &= \{P(m) \cap CP | m \in ES\} \end{aligned}$$

The algorithm should be revised as follows:

```
1. create list of hyperedges $H$
2. create list of extra hyperedges $EH$
3. create a tree and set the smallest hyperedge in $H$ as root
4. for any node in tree do following steps
    4.1 mark all hyperedges in $H$ and $EH$ hit by last selected node as settled
    4.2 IF there is no unsettled hyperedge in $E$
        4.2.1 add the list of selected vertices up to that node to the list of
              solutions
    4.3 ELSEIF the number of selected vertices up to that node is $\leq K$ AND
        the number of settled hyperedges in $EH$ is $\leq f$
        4.3.1 choose smallest unsettled hyperedge $h$ and mark it settled
        4.3.2 branch on every vertex contained in $h$ by adding it to the list of
              selected vertices
    4.4 mark as dead end
```

2.3.3 The algorithm: spurious masses

The algorithm should be altered for spurious masses in a way that provides the possibility of a hyperedge being considered settled without its being hit with a selected vertex. This leads to introduction of nodes where a hyperedge is chosen and marked spurious where no action is taken over its vertices and just a branch is made by removing that hyperedge from set of hyperedges.

```
1. create list of hyperedges $H$
2. create a tree and set the smallest hyperedge in $H$ as root
3. for any node in tree do following steps
    3.1 mark all hyperedges hit by last selected node as settled
    3.2 IF all hyperedges are either settled or marked spurious
        3.2.1 add the list of selected vertices up to that node to the list of solutions
    3.3 ELSEIF the number of selected vertices up to that node is $\leq K$ AND number of spurious hyperedges is $\leq g$
        3.3.1 choose smallest unsettled hyperedge $h$ and mark it settled
        3.3.2 branch on every vertex contained in $h$ by adding it to the list of selected vertices
        3.3.3 IF number of spurious hyperedges is $< g$ THEN mark $h$ spurious and branch (without selecting any vertices)
    3.4 mark as dead end
```

2.3.4 Avoiding duplicate solutions

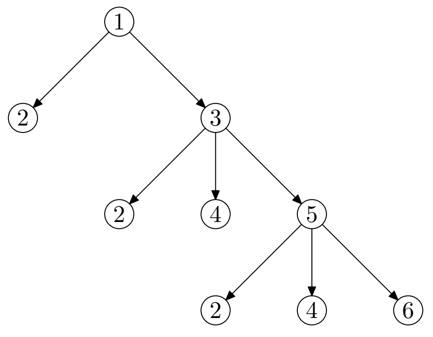
The easiest way of suppressing duplicate solutions is a pair-wise comparison after the main algorithm. In the implementation of algorithm a set is also passed from a parents to children in every branching that contains all the vertices that has already been checked by previous branchings. For instance, for a set of proteins numbered 1 to 6, a set of hyperedges can lead to a tree like Figure 2.1(a). A fast (but not the best) way is for each node to remember all the checked vertices to its left and the checked vertices passed from its parent. In this case, on branching on node 3, examining vertex 2 is redundant. Therefore, the algorithm only branches on 4 and 5 and passes 2 as a checked vertex to node 5. This node now knows that 2 and are already tried so it just checks node 6 and passes nodes $\{2, 4\}$ to node 6.

This is not optimal, because one should pass on all combinations of vertices that are already checked.

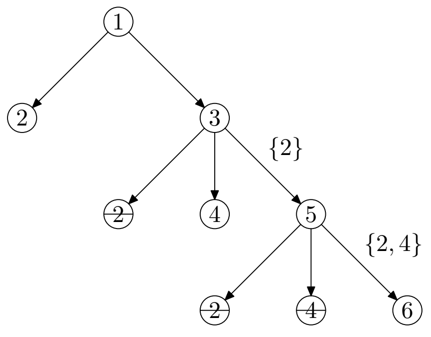
2.4 Preprocessing and initial candidates

2.4.1 Creating list of hyperedges

Initially, the idea used for creating the list of hyperedges was using the table for masses. In error-free case, one can take union of all proteins associated with



(a) Original tree



(b) Reduced tree

Figure 2.1: Simple duplicate solution avoidance technique.

every mass in the sample, then selects those that their spectrum is completely contained by the sample. This can even expanded for missing masses case, where one can also add all proteins that have no more than f fragment masses. However, this approach turned to be costly because, there are several masses that are associated to more than 130,000 proteins (reduced set of proteins contains around 178,000 proteins). So it's quite likely (probability increasing by the number of proteins in mixture) that they appear in the sample mixture. In several tests, it was realised that the final candidate set covers more than 85% of proteins. So the cost for checking the remaining proteins was much less than compiling the initial set of candidate proteins. Therefore, in the implementation, a search is carried out over entire set of proteins.

The checking against each protein is also a costly routine. In the beginning, STL function `set_intersection` was employed to obtain the size of intersection of the sample and spectrum of each protein and the size of it was used to check if the difference set is small enough to be eligible as a candidate protein or not. It turned out that it's also computationally expensive and in the error-free case the running time of this part was dominant[4]. A separate routine `iseligible` (Appendix B.1) was later developed that terminated the routine any time it concluded that there are more than f in the protein spectrum that are not in the sample spectrum. This routine reduced the running time for the checking more than 90%[5]. Now, the running time of this part is comparable in magnitude to other parts.

Another restricting conditioned which is employed is the fact that for cardinality of $A \backslash B$ to be less than f, it is necessary that $|A| - |B| \leq f$. So the checking is only tried if this condition holds. This reduces the running time of checking part roughly 4%.

Starting with small number of branches improves the algorithm, so the list of hyperedges is sorted by size giving priority to smaller hyperedges.

[4]In average more than 95% of running time.

[5]In one extreme case the time was reduced from 5.94 seconds to 0.556 seconds

2.4.2 Proteins with identical fingerprint

While digesting proteins with a certain enzyme, it's quite likely to find two distinct proteins that result in identical fragment mass spectrum (fingerprint). To reduce the effect of this phenomenon, more than one enzyme is used for digestion and the results of database search on fragment masses produced by each enzyme are compared and intersection of results from different searches are considered. In this thesis, a program called `unique` (see A.2) is developed that removes all duplicate copies of a protein. The code may also create a cross-reference table that groups the equivalent proteins. This file can be used in generating of reports to indicate other possible solutions.

2.4.3 Unique fragment masses and Duplicate hyperedges

In some cases a fragment mass belongs only to one single protein in database. In case of error-free and missing masses problems, such proteins belong to all solutions. In these cases the node corresponding to that protein is added to the set of selected nodes in the root of the search tree and all hyperedges that contain this node are considered settled and removed from initial set of hyperedges. The number of vertices chosen this way for mixture m are denoted $N_{iv}(m)$. Removing all hyperedges that are hit by these vertices from the set of sample fragment masses yields a new set of unsettled hyperedges. The cardinality of this set is denoted by $N_{uh}(m)$ for a given mixture m. Note that this reduction cannot be used for the case of spurious masses where instead no node is pre-selected in root because the very unique fragment mass can be among the spurious masses in sample mixture.

Duplicate hyperedges are also removed except for the case of spurious masses. Each copy of a hyperedge can represent a spurious mass.

2.5 Simulation

A range of parameters and measures are simulated in this project. First of all a time-complexity survey on real data is carried out. Additionally, information regarding the distribution of size of mixtures for a given number of proteins, distribution of kernel size and number of solutions in presence of different kind of errors are investigated. Since the set of all mixtures is a huge space to simulate, a small subset is sampled for analysis.

The simulations are all carried out on Chalmers PC-cluster `ada`. Jobs were run as single-thread tasks. Each node is equipped with[6]:

- 4 Xeon 5160 (Woodcrest) 3 GHz cores (dual dual core)
- 4GB RAM, 1GB / core
- 100GB free local storage

[6]Retrieved from `http://www.c3se.chalmers.se/index.php/Hardware_Ada_/_Kal`

2.5.1 Sampling and average values

There are 179,748 distinct proteins in the database and even considering all two-protein mixtures means around 16 billion mixtures let alone five-protein mixtures that their number exceeds $1.5 \cdot 10^{24}$. In first stage of simulation we build 100 million mixtures for mixtures of up to 50 proteins. Later, at most 10 mixtures are chosen for any given mixture size. However, size of generated mixtures are saved to compile an approximate of distribution of mixture size for any given number of proteins.

In this manuscript the term *average* is used whenever the value mentioned is the average all data points that share one mixture size. For instance, average $N_{cp}(m)$ means the mean of number of candidate proteins for all samples that have mixture size m.

2.5.2 Time measurement

The time spent on three parts of the algorithm is measures:

- T_{cp}, Time spent on comparing sample fragment masses with fingerprints of proteins and creating set of candidate proteins CP;
- T_{he}, Time spent on creating the list of hyperedges;
- T_b, Time spent on creating the search tree and enumerating the solutions;

All times are measured with microsecond accuracy.

Chapter 3

Results and discussion

In implementation of the code initialization of tables for mass fragments and proteins are executed once and the execution time is measured for the test mixtures not considering the initialization time. All regressions are carried out using MATLAB functions `polyfit` and `nlinfit`.

3.1 Distribution of number of fragment masses

The number of distinct fragment masses produced by trypsin digestion varies between 1 and 768. There are 400 different values for this number. The distribution of number of fragment masses is depicted in Figure 3.1 where 99% (90%) of proteins digest into less than 141 (64) fragments. It should be noted that there are 716 proteins that result in one single fragment.

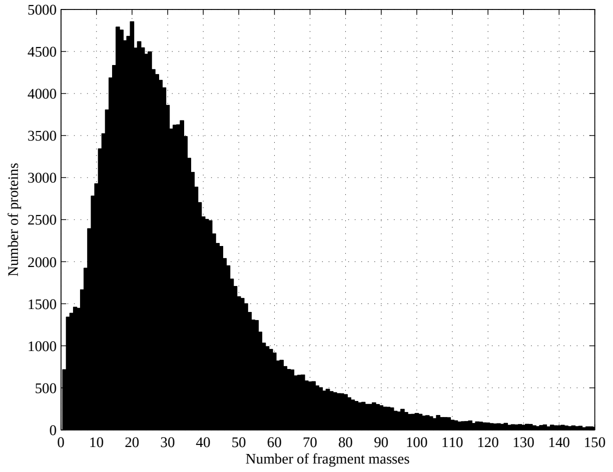


Figure 3.1: Distributions of number of fragment masses in protein database. Note that the horizontal axis is truncated at 150 since there are less than 25 proteins for a given number of fragment masses above 150.

This can lead to several alternative solutions in case another protein in sample mixture shares that fragment mass. The most common outcome is 20 which accounts for 4855 proteins (2.52% of all proteins).

3.2 Distribution of mixture size

Distribution of mixture size is calculate over 100 million random mixtures. To ensure a good coverage in generating random mixtures, constituent proteins of last random mixture were removed and next mixture is taken from remaining proteins. Whenever the list of proteins were exhausted, it was reset to the initial set. Let $D(m,n)$ be the number of mixtures of size m that can be produced by mixing n proteins divided by total number of mixtures. What we achieve is $\tilde{D}(m,n)$, an approximation of original distribution.

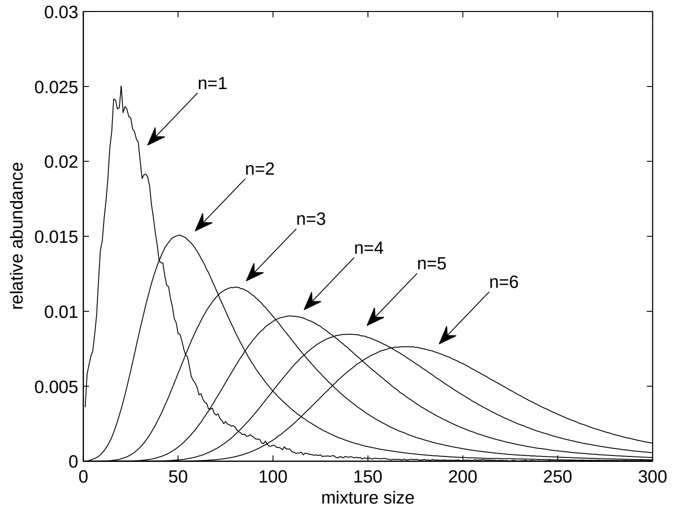


Figure 3.2: Relative abundance of mixture sizes for mixtures containing up to six proteins.

Note that the distribution for case of one protein is taken directly from database. The relative abundance for mixtures of up to six proteins is depicted in Figure 3.2. The distribution functions get wider and their maximum becomes smaller as the number of proteins grows. The graphs for mixtures of up to 50 proteins is depicted in Figure 3.3.

The maximum abundance, $A(n)$, decreases as number of proteins, n, increases. It also resembles a ration function of two quadratic functions. We try to estimate $A(n)$ using

$$\hat{A}(n) = K_A \frac{n^2 + a_1 n + a_0}{n^2 + b_1 n + b_0} = K_A + K'_A \frac{n + a'_0}{n^2 + b_1 n + b_0}$$

Nonlinear regression yields

$$K_A = 0.001857, \quad a_1 = 47.256, \quad a_0 = 52.055, \quad b_1 = 12.890, \quad b_0 = 0.983$$

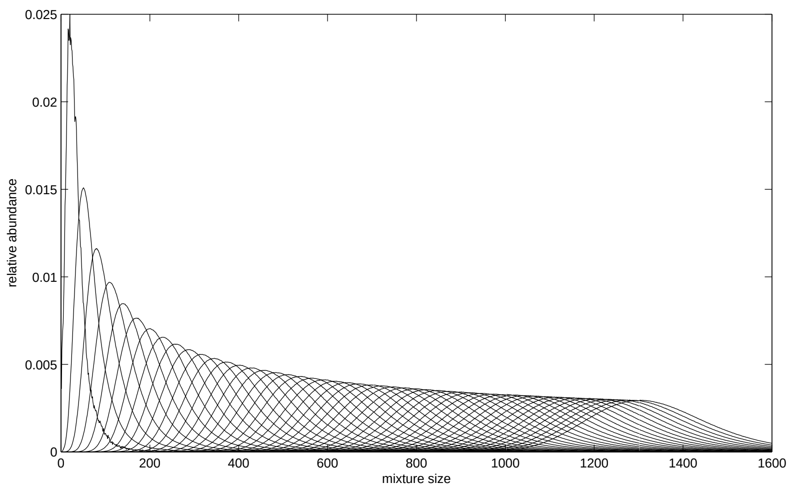


Figure 3.3: Relative abundance of mixture sizes for mixtures containing up to 50 proteins.

or

$$K_A = 0.001857, \quad K'_A = 0.063833, \quad a'_0 = 4.39598$$

The roots of denominator are $n_0 = -12.81374$ and $n_1 = -0.076743$. It's surprising close function $\hat{A}(n)$ follows values of $A(n)$, see Figure 3.4(a). In fact the error is always in order of 10^{-5}. One of the roots of denominator is close to 0 (in comparison to the values n takes: $1, 2, \ldots, 50$). So, one can assume a quadratic function of form $n(n + n_1)$ as denominator and simplify the regression. This results in

$$K_A = 0.001968, \quad K'_A = 0.05567, \quad a'_0 = 2.9253, \quad b_1 = 8.472, \quad b_0 = 0$$

This still is a very good approximation of data with errors in order of 10^{-4}

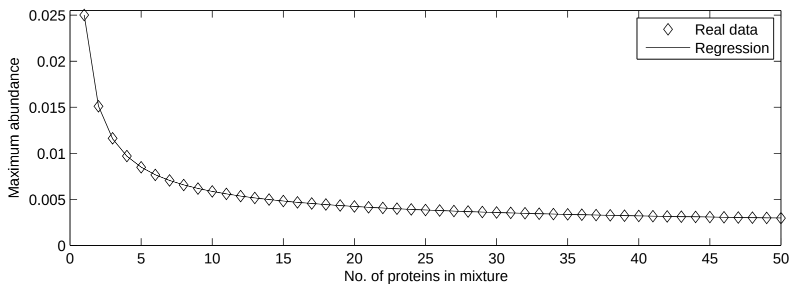
The mode of the distribution (or the most abundant mixture size), $S(n)$, shows a tendency to grow linearly with number of proteins though a better fit can be found with a higher-order polynomial regression, the decrease in error is negligible. Regression using polynomials of degrees higher than 3 show very small values for coefficients of high-degree terms. So a cubic function $\hat{S}(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3$ is employed. Regression yields

$$a_0 = -11.9743, \quad a_1 = 31.2045, \quad a_2 = -0.1315, \quad a_3 = 0.0006677$$

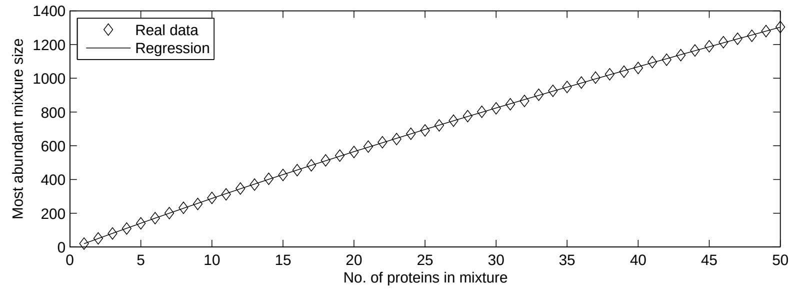
Still, a_3 is a substantially small value and a quadratic function can be considered. The functions fits the data as depicted in Figure 3.4(b).

3.2.1 The mixture size follows log-normal distribution

The non-symmetrical distribution of mixture sizes show a similarity to distributions like negative binomial or lognormal. The hypothesis testing using



(a) $A(n)$ and $\hat{A}(n)$



(b) $S(n)$ and $\hat{S}(n)$

Figure 3.4: Maximum relative abundance and most abundant mixture size vs. number of proteins in mixture

Kolmogorov-Smirnov method validated that the distributions are following a threshold lognormal distribution[1]. For any given p the size of mixture will have a minimum. That is, by mixing p we cannot get any arbitrary small mixture size. The rough lower-bound is p, simply because one can choose p proteins that each contain one mass unique to itself and not shared with other $p-1$. Therefore, the existence of a threshold is then inevitable.

To estimate the parameters there are several methods proposed including Maximum-Likelihood, LMSE over PDF and LMSE over CDF. Here we use LMSE over PDF to find the closest distribution. The results acquired by ML and CDF-LMSE methods emphasize on good tail match and therefore to some extent miss the main peak. Since in our simulation we are doubtful about the accuracy of results on tails, we chose PDF-LMSE.

To illustrate this the case of $p = 22$ is taken as an example and results are illustrated in Figure 3.5. The LMSE fitted distribution results in a threshold lognormal with parameters

$$\mu = 6.16703, \quad \sigma = 0.21242, \quad \theta = 161.02$$

The threshold of 161 shows that the lognormal distribution is shifted 161 places. This can also be an indication that no mixture of less than size 161 can be produced by 22 proteins. The thresholds calculated for $1 \leq p \leq 50$ can be seen in Figure 3.6. The threshold is a negative value for $1 \leq p \leq 3$. This can

[1]also known as three-parameter lognormal

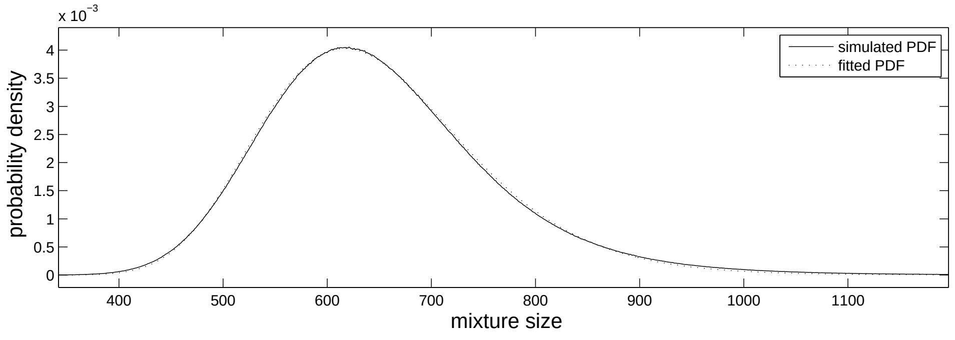


Figure 3.5: Comparison of simulated and lognormal-fitted PDF of mixture size for $p = 22$. The x-axis is truncated to enhance visibility of the slight difference.

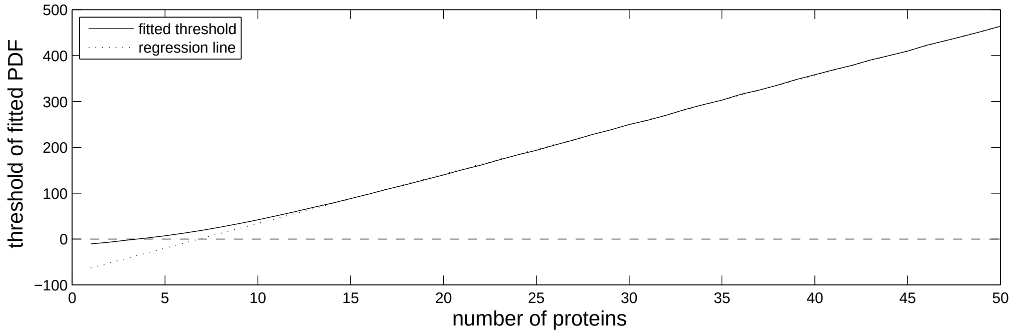


Figure 3.6: The trend of threshold of fitted three-parameter lognormal distribution for $1 \leq p \leq 50$.

be interpreted as a truncated lognormal distribution. This can be easily seen in Figure 3.2 for the case of $p = 1$. For $p > 12$ the threshold is strongly linear. The slope is 10.7665 and y-intercept id -73.849. The difference from the linear regression for $p \leq 12$ shows a strong logarithmic relation.

3.2.2 Estimating number of proteins based on mixture size

The distributions achieved by simulation can be used as a probability density function. Let M and N be two random variables denoting number of fragment masses and proteins in a random mixture. In a probabilistic fashion

$$P(M = m|N = n) = D(m, n)$$

The problem of estimating number of proteins given number of fragment masses translates to finding $\tilde{n}$ so that $P(N = \tilde{n}|M = m)$ is larger than other values of n, i.e.

$$\tilde{n} = \arg\max_n P(N = n|M = m)$$

Finding a solution for this maximum-likelihood problem requires some *a priori* knowledge about the distribution of N. So given this distribution and using Bayesian principle of

$$P(N = n|M = m) = \frac{P(N = n)}{P(M = m)} P(M = m|N = n) = \frac{P(N = n)}{P(M = m)} D(m, n)$$

Since we are concerned about a fixed m, what we need is $P(N = n)$ to find the most probable number of proteins.

3.3 Error-free case

All sample mixtures with introduction of no errors are analysed to determine the performance of the algorithm. To determine the effect of K on performance, all mixtures of up to 20 proteins are analysed for all values of $1 \leq K \leq 20$. This gives an indication of how costly would be if one chooses a K larger than real number of proteins.

3.3.1 The effect of uniqueness check

In case of error-free analysis, the checking for unique fragment masses significantly reduces the search space. The results show that a huge proportion of mixtures result in N_{iv} close to the number of proteins that constituted the mixture in first place. That means only a small number of vertices are left to be identified when the search tree is constructed. It is evident that when $N_{iv} = p$ all hyperedges are hit and $N_{uh} = 0$, no regularity is noticed in case of $N_{iv} < p$. Mixtures of up to 15 proteins, are either totally identified or only one, or in are cases two, proteins are left. Naturally the proportion of number of mixtures totally identified by uniqueness check to total number of mixtures reduces by increase of number of proteins in the mixture. This reflects the new possibilities that arises by combination of several proteins that, in average, makes it less likely for a given fragment mass to be unique. The relative frequency of mixtures that result in $N_{iv} = p - i$, where p is the number of proteins in the mixture and $0 \leq i$ in an arbitrary integer is shown in Figure 3.7. In simulation of analysis of mixtures of up to 50 proteins no case of $i > 6$ where encountered. This doesn't necessary mean there is no such possibility, but the probability of it happening is certainly low.

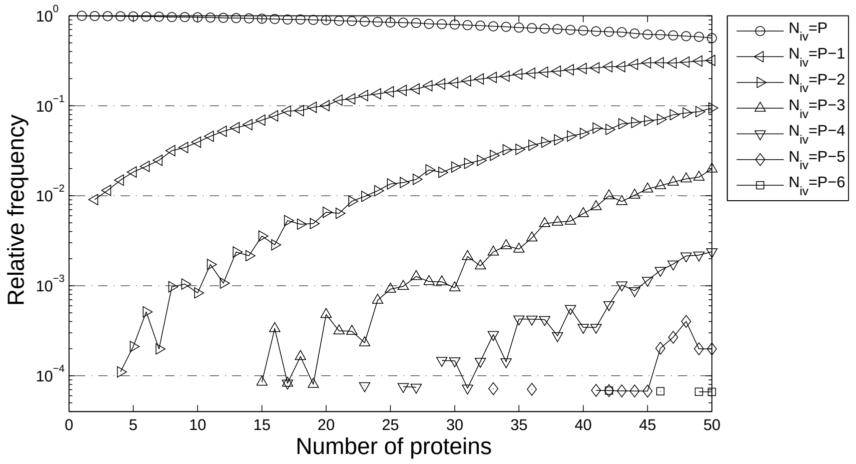


Figure 3.7: Relative frequency of mixtures resulting in a fixed number of initial vertices, N_{iv}, for mixtures produces by up to 50 proteins. P denotes number of proteins.

The size of the search tree depends, among others, on number of remaining

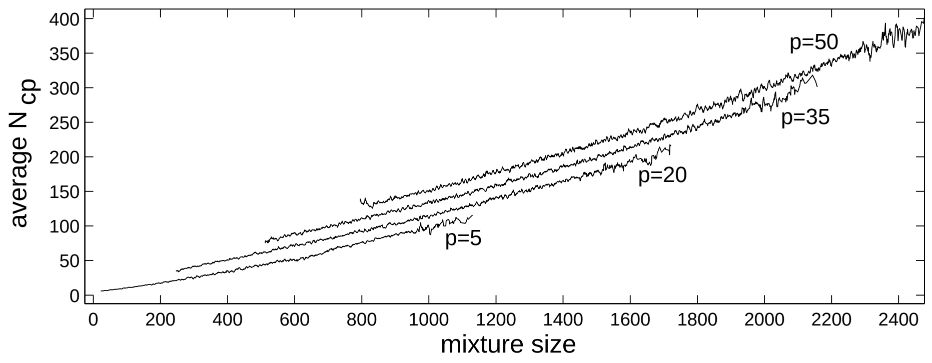


Figure 3.8: Average number of candidate proteins increases by increasing values of p for any given mixture size. The curves are smoothed by a moving average filter of length 5.

vertices and number of remaining unsettled hyperedges. In fact, these numbers should be considered as starting point of the algorithm, so in analysis of the algorithm we consider $p - N_{iv}$ and N_{uh} instead of p and m as parameters controlling the complexity of branching part of the algorithm.

3.3.2 Number of candidate proteins

Number of candidate proteins, N_{cp}, determines the number of vertices in the hypergraph and the time required for creating CP, T_{cp}, heavily depends on this number. The number of candidate proteins is a function of:

- size of protein database and distribution of fragments in protein spectra
- number of masses in sample fragment spectrum
- number of proteins in sample mixture

In this investigation we are focused on a fixed database. Therefor the dependency on database size is not investigated. The fact that N_{cp} depends on number of proteins is not obvious in first glance. One can claim that no matter how many proteins where mixed to create the mixture, the number of fragment masses will dominate N_{cp}. However, observations in this investigation revealed that for mixtures of a given size, those that are produced by larger number of proteins lead to a larger number of candidate proteins hence larger T_{cp}. However, this effect is very weak for some masses. This can be easily seen in Figure 3.8 where the average N_{cp} is drawn for four equally distanced number of proteins. For instance, for $m = 1000$, average number of candidate proteins for $p = 5, 20, 35$ and 50 are 101.7, 110.7, 135.5 and 157.5 respectively.

The average N_{cp} for mixture size of 1000 is illustrated in Figure 3.9. A strong linear dependency is observed and verified for all mixture sizes. The linearity is much stronger for $p > 15$. Introduction of a second-order term causes a slight reduction in error. In case of $m = 1000$, a second-order polynomial reduces SSE from 421.74 to 420.67 where R^2 statistic increases from 0.94038 to 0.94053

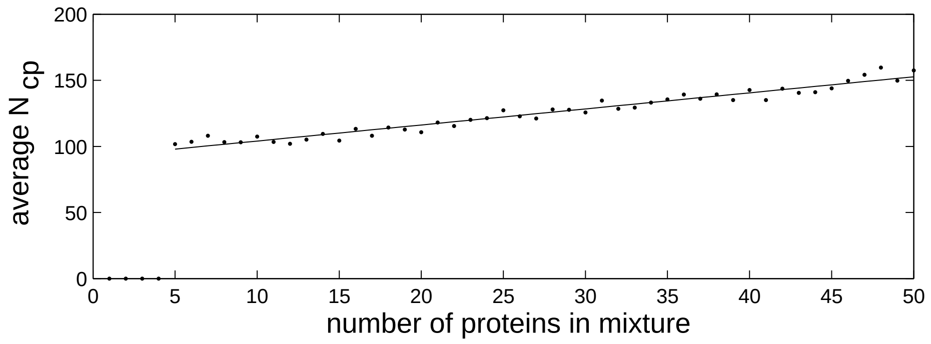


Figure 3.9: Linear trend of average N_{cp} for $m = 1000$ with respect to number of proteins in mixture. Note that no mixture os size 1000 were produced for $p \leq 4$ and these values are not included in regression model.

indicating a very good fit for linear regression. So we conclude that for a fixed mixture size

$$N_{cp} = O(p) \tag{3.1}$$

Now we just try to see how fast N_{cp} grows by m. We try polynomial and exponential regressions to achieve a fitting. We study the case $p = 35$. Several candidates are tested and results are tabulated in Table 3.3.2. . For polynomial

function	RMSE	adj-R^2	comments
$f_1(m) = a_1m + a_0$	7.3493	0.98561	
$f_2(m) = a_2m^2 + a_1m + a_0$	6.1356	0.98997	
$f_3(m) = a_3m^3 + a_2m^2 + a_1m + a_0$	6.1297	0.98999	
$f_4(m) = a_4m^4 + a_3m^3 + a_2m^2 + a_1m + a_0$	6.1264	0.99000	
$f_5(m) = kb^m$	8.2445	0.98189	b=1.00075935
$f_6(m) = kb^m + a_0$	6.1293	0.98999	b=1.00032303
$f_7(m) = kb^m + a_1m + a_0$	6.1304	0.98998	b=1.00049563
$f_8(m) = kb^m + a_2m^2 + a_1m + a_0$	6.1333	0.98998	b=1.00027993

Table 3.1: Goodness-of-fit parameters for several functions fitted to N_{cp} with respect to m for mixtures produced by 35 proteins.

functions a quadratic functions shows a strong relevance since the RMSE does not reduce significantly for higher-degree polynomials. For the case of exponential and mixed polynomial-exponential functions $f_6(m)$ shows the best RMSE. It should be noted that $f_7(m)$ and $f_8(m)$ lead to very wide 95% confidence intervals. So we are left with two candidates:

$$\begin{aligned} f_2(m) &= a_2m^2 + a_1m + a_0 \\ f_6(m) &= kb^m + a_0 \end{aligned}$$

where

$$a_2 = 0.00002164, \qquad a_1 = 0.0782, \qquad a_0 = 33.40$$

$$k = 271.8, \qquad b = 1.00032303, \qquad a_0 = -242.0$$

Functions $f_2(m)$ and $f_6(m)$ are both very good candidates and current amount of information might not provide enough evidence to prefer one to the other. Let $e(m) = |f_2(m) - f_6(m)|$, one can see that $\max\{e(m)\} < 1.2$ and average of $e(m)$ is 0.15143 where the range of these function covers the interval of [75,318]. However, the base of exponential function is very close to 1 and the domain we are concerned is rather limited and the range of mixture sizes covered in this investigation provide a relatively good estimate of the real range. Therefore, we choose the quadratic function over exponential one because of the computational robustness it introduces later stage. Similar investigation for $1 \leq p \leq 50$ shows similar behaviour with slight deviations for $1 \leq p \leq 5$. In these cases for mixture sizes smaller than a threshold, there is a strong linear trend that later changes to a quadratic. However, the fitting with quadratic function is still reasonable in those cases and we can deduce that for a fixes number of proteins

$$N_{cp} = O(m^2) \tag{3.2}$$

combining this with (3.1) yields eithr

$$N_{cp} = O(pm^2) \tag{3.3}$$

or,

$$N_{cp} = O(p + m^2) \tag{3.4}$$

We define a bivariate polynomial function, $f_{cp}(p, m)$, and fit it to the whole domain of available values for N_{cp}

$$f_{cp}(p, m) = (a_{12}p + a_{02})m^2 + (a_{11}p + a_{01})m + (a_{10}p + a_{00})$$

The fitting shows a very close fitting with RMSE=6.2333 and $R^2 = 0.99419$. The coefficients can be presented in a matrix form where a_{ij} is in row i, column j

$$A_{cp} = \begin{pmatrix} -4.6446 & 7.742 \cdot 10^{-2} & 1.7709 \cdot 10^{-5} \\ 1.1342 & -6.3773 \cdot 10^{-5} & 1.4706 \cdot 10^{-7} \end{pmatrix}$$

Now we remove one term from the function to see how different terms affect the final RMSE. The result is summarized in Table 3.2. All terms, except a_{11} show a significant change in final error.

	removed term						
	none	a_{12}	a_{02}	a_{11}	a_{01}	a_{10}	a_{00}
RMSE	6.2333	6.2631	6.5139	6.2344	7.8035	7.0346	6.2711
Δ RMSE	0	0.0298	0.2806	0.0011	1.5702	0.8013	0.0378

Table 3.2: Effect of removal of each term in $f_{cp}(p, m)$ on overal RMSE

Removing least influential terms, $a_{11}pm + a_{12}pm^2$, leads to a fitting with RMSE=6.3416 (Δ RMSE=0.1083) which is definitely unacceptable[2]. Hence, we ignore the possibility of (3.4) and find $N_{cp} = O(pm^2)$ more relevant to the data in hand.

[2]Note that RMSE is root mean of squared error over 69,591 data points and 0.1 deviation in RMSE is around 94709 deviation in SSE.

3.3.3 Time complexity

The time complexity as mentioned in section 2.5.2 is divided into three main constituents.

Creating CP

Building CP requires comparison of sample fragment spectrum with spectra of all proteins in database. The required time is a function of

- size of protein database
- number of candidate proteins which in turn is a function of number of masses and proteins in the mixture.

Database size is fixed in this, so we just consider the effect of N_{cp}. Figure 3.10 shows the relationship of T_{cp} and N_{cp} for $p = 35$. It is strongly linear. Introduction of a quadratic term only reduces RMSE from 18397 to 18386.5 .

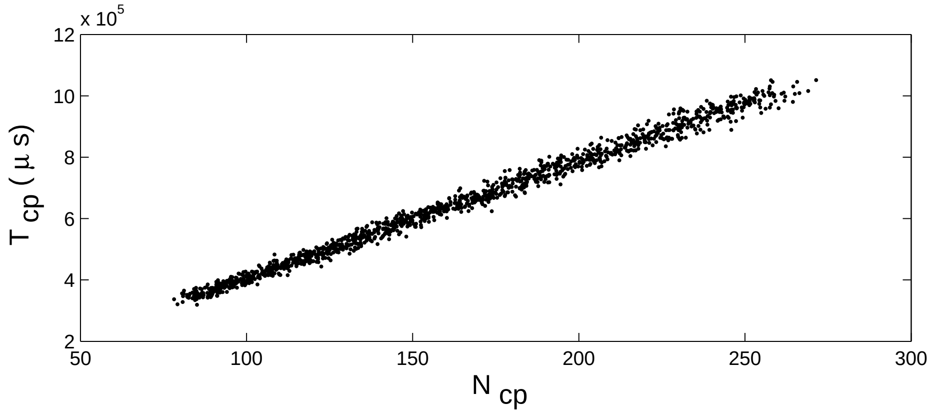


Figure 3.10: T_{cp} vs. N_{cp} for $p = 35$.

Similar relationship can be found for any $1 \leq p \leq 50$ with a slope of 3600 ± 200. Therefore,

$$T_{cp} \leq 3900 N_{cp}$$

or,

$$T_{cp} = O(N_{cp}) = O(pm^2) \tag{3.5}$$

Creating HE

The time required to create the list of hyperedges for mixtures of 35 proteins is illustrated in Figure 3.11. The relationship is not strictly linear. In fact, adding a term containing square-root of m, i.e. $a_2 m + a_1 m^{0.5} + a_0$ fits a bit better. However, the share of the second term in total value of the function is negligible for most values of m therefore a linear function is a good enough representation, especially for larger values of mixture size. Hence,

$$T_{he} = O(m) \tag{3.6}$$

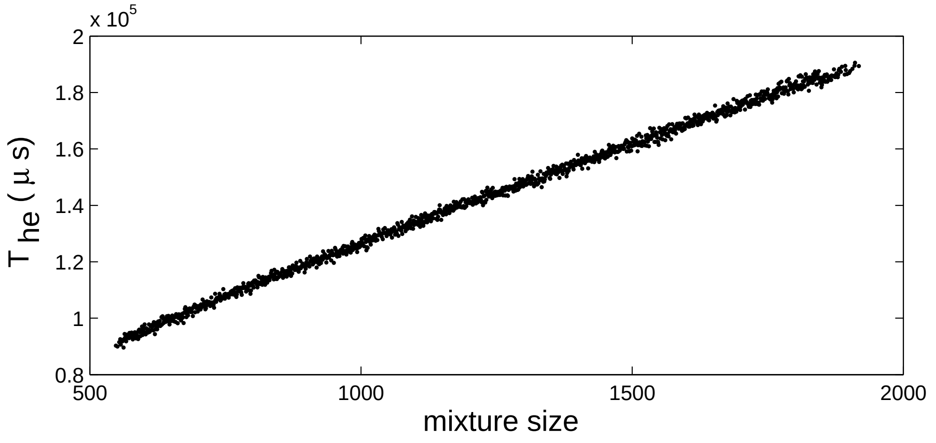
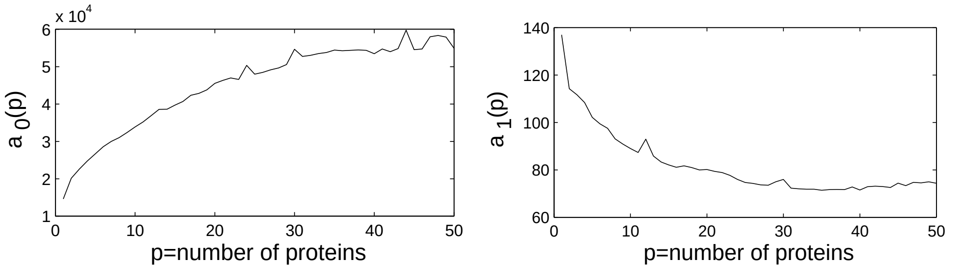
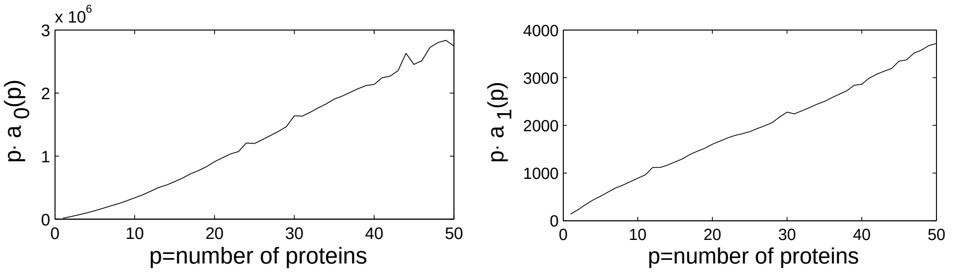


Figure 3.11: Time for creating H vs. mixture size for $p = 35$.

Let $T_{he}(p, m) = a_1(p)m + a_2(p)$, we carry out a linear regression for all $1 \leq p \leq 50$. Values obtained for $a_0(p)$, $a_1(p)$, $p \cdot a_0(p)$ and $p \cdot a_1(p)$ are depicted in Figures 3.12(a)-3.12(d).



(a) (b)



(c) (d)

There is a linear trend in $p \cdot a_0(p)$ and each can be fitted to a linear function hence

$$a_0(p) = O(p^{-1}), \qquad , a_1(p) = O(p^{-1})$$

or by combining with 3.6,

$$T_{he}(p, m) = O(p^{-1}m) = O(m) \tag{3.7}$$

Solving the regression equations for numerical values yields

$$T_{he}(p,m) = (68.567 + 190.86p^{-1})m + (63908 - 357180p^{-1})$$

Creating search tree

It is important to note that in error-free case the branching time is a very small part of total time. In average it takes no more than 200μs to finish where N_{cp} and T_{he} are in order of 10^5 μs. However, studying its behaviour is important for comparison to erroneous cases. The branching time is highly influenced by N_{iv} and for a given N_{iv} it is tightly concentrated around a mean value. This mean value however shows a power-law relation with $p - N_{iv}$, i.e. the number of vertices sent to branching algorithm. However, since there are usually very few samples for larger values of $p - N_{iv}$ the accuracy of it cannot be verified. However an exponential regression tields

$$T_b \approx 65.39 \cdot 2.5995^{p-N_{iv}} = O(2.6^{p-N_{iv}}) \tag{3.8}$$

To make a better assessment of how fast T_b grows, instead of regression, we try to find the smallest value of b that $T_b/b^{p-N_{iv}}$ is monotonically decreasing. The result is depicted in Figure 3.12[3] and it's obvious that it's a big overestimating (for $p = 35$, it returns 4 instead of 2.6 we achieved), however, it reveals that regardless of p

$$T_b = O(5.2^{p-N_{iv}})$$

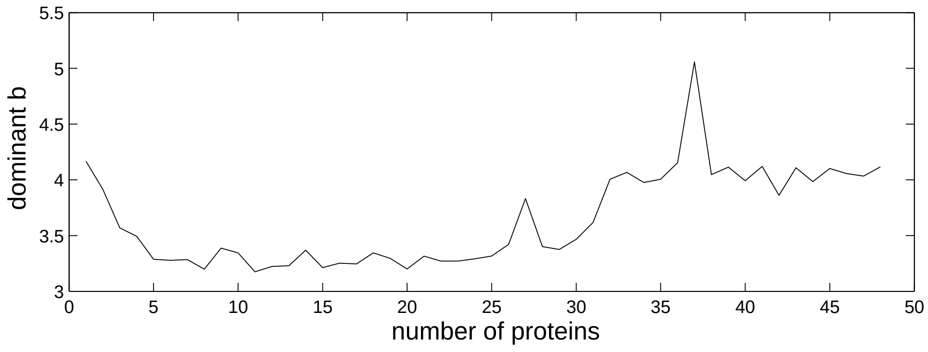


Figure 3.12: Dominant exponential base for T_b

Figure 3.13 shows the average T_b for different values of N_{iv} in case of $p = 35$. Also it is important to note that how often those cases happen. This issue has addresses under effect of uniqueness check.

[3]Note that results for $p = 49$ and 50 are missing because of some cases of extremely large search trees that significantly deviated average values of T_b for these cases. A similar but less significant case can also be seen for $p = 37$

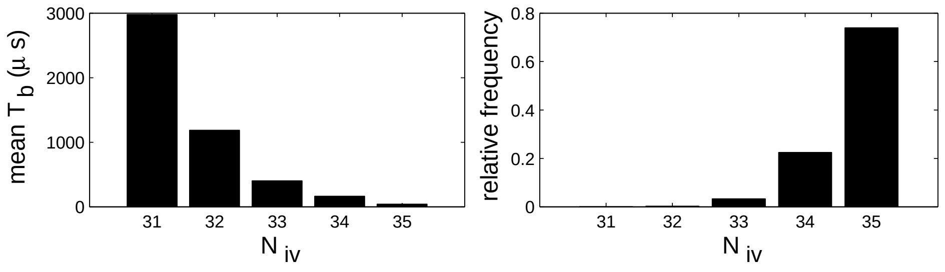


Figure 3.13: Mean T_b for different values of N_{iv} and relative frequency of mixtures leading to that value of N_{iv}.

We approximate, $\bar{T_b}$, the mean of T_b by taking expected value of T_b using the distribution function we approximated earlier, i.e.

$$\bar{T_b}(p) = E_m\{T_b(p,m)\} \approx \sum_m D(m,p)T_b(p,m)$$

Figure 3.14 shows the $\bar{T_b}$ and its logarithm for all $1 \leq p \leq 50$. An exponential trend is recognisable from logarithmic plot.

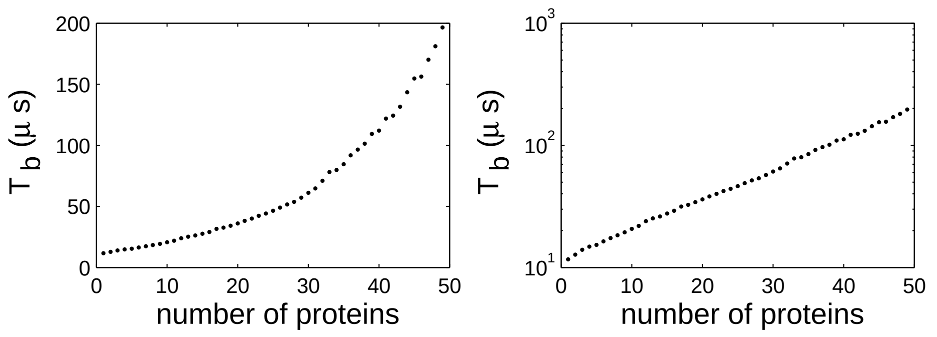


Figure 3.14: Mean T_b and its logarithm vs. number of proteins.

An exponential regression by fitting a function of form ab^p yields

$$a = 11.4, \qquad b = 1.0543$$

$R^2 = 0.9988$ also confirms that the linearity is indeed very strong. Hence,

$$\bar{T_b} = O(1.055^p) \tag{3.9}$$

Overall time complexity

Overall time-complexity achived by this simulation for error-free case can be summarised as

$$T_{all} = T_{cp} + T_{he} + T_b = O(pm^2) + O(m) + O(5.2^{p-N_{iv}}) = O(pm^2 + 5.2^{p-N_{iv}})$$

3.4 Missing masses

The algorithm for missing masses is tested over mixtures containing 1 to 20 proteins with $1 \leq f \leq 10$. The number of missing masses in mixtures was not initially limited to 10, but in cases of more than 10 missing masses there were several occasions that the search tree became too large to be accommodated in the available memory on the systems so here the results for $f \leq 10$ are studies.

3.4.1 The effect of uniqueness check

The effect of uniqueness check is summarized and compared to error-free case in Table 3.3. The maximum value of $p - N_{iv}$ dominates T_b. However, the maximum value is not reached so often. For instance for the case of $f = 6$, the behavior of this value is depicted in Figure 3.15.

		f										
		0	1	2	3	4	5	6	7	8	9	10
	1	0	1	1	1	1	1	1	1	1	1	1
	2	1	2	2	2	2	2	2	2	2	2	2
	3	1	3	3	3	3	3	3	3	3	3	3
	4	2	2	3	3	4	4	4	4	4	4	4
	5	2	3	4	4	5	5	5	5	5	5	5
	6	2	3	3	4	4	6	6	6	6	6	6
	7	2	3	3	5	5	6	6	7	7	7	7
	8	2	3	4	4	5	5	7	7	8	8	8
	9	2	4	5	5	6	7	7	8	9	9	9
	10	2	4	5	7	7	8	9	9	9	10	10
p	11	2	4	6	6	7	8	9	9	10	10	10
	12	2	5	6	6	7	7	9	10	11	11	12
	13	2	5	5	6	8	9	10	10	11	12	12
	14	2	5	6	7	9	9	10	12	13	13	13
	15	3	4	6	6	9	11	12	13	13	13	14
	16	3	5	6	8	9	10	11	13	13	14	16
	17	4	5	6	7	8	9	12	14	15	16(1)	16(1)
	18	3	5	6	8	9	11	14	14	15	15(1)	17(1)
	19	3	6	6	8	9	12	14	15	16(1)	16(1)	17(1)
	20	3	5	7	7	10	12	13	15	17	18(2)	18(3)

Table 3.3: The maximum value of $p - N_{iv}$. In most cases the minimum value is zero, for remaining cases the minimum value is indicated in parenthesis.

Comparing Figure 3.15 with Figure 3.7 reveals a similar trend of decrease in share of $p - N_{iv} = 0$. An initial increase in relative frequency of all subsequent cases are observed however the trend is very slow in error-free cases and simulation for $p \leq 50$ does not provide enough evidence to support that a peak also happens in error-free case or not. However, a similar behavior is observed for all $1 \leq f \leq 10$ where increase in f moves the peak points to the left. Obviously, increasing f worsens the improvement achieved by uniqueness check.

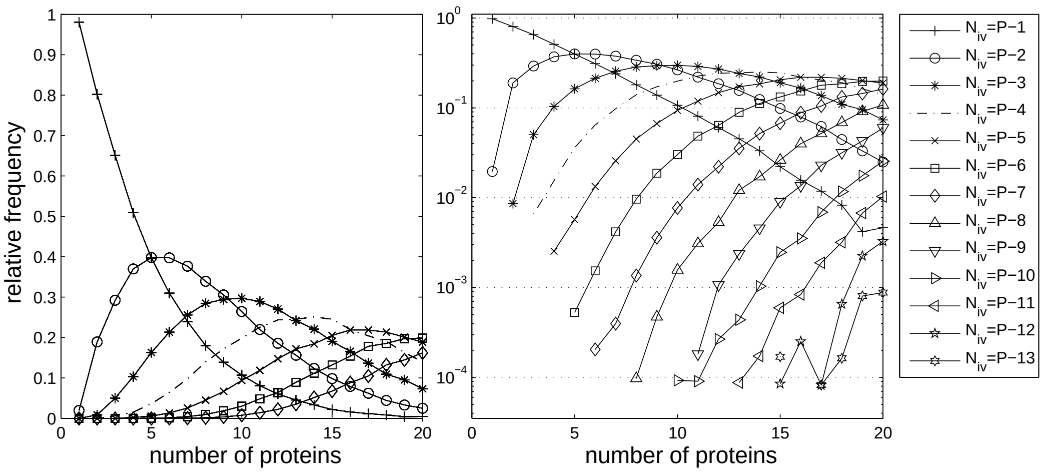


Figure 3.15: Relative frequency of mixtures with six missing masses resulting in a fixed number of initial vertices, N_{iv}, for mixtures produces by up to 20 proteins. P denotes number of proteins. In linear plot only first 7 curves are presented.

3.4.2 Number of candidate proteins

To illustrate the dependency of N_{cp} on number of proteins we fix the mixture size, m, and draw N_{cp} for different values of f with respect to p. The results show a difference from error-free case. In existence of missing masses N_{cp} is almost independent of p. This is illustrated in Figure 3.16 for $m = 500$ and $m = 1100$. Similar results are confirmed for all values of m. The figures are drawn in logarithmic scale to provide visibility for error-free result. Therefore, we consider N_{cp} as a function of f and m.

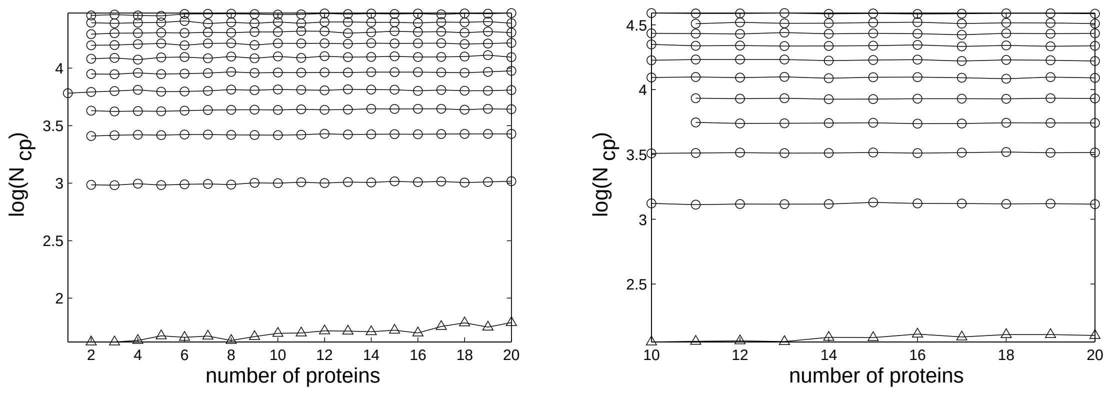
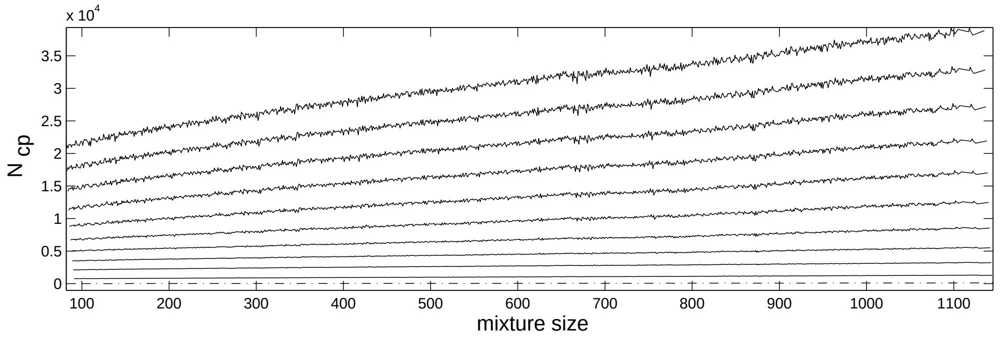


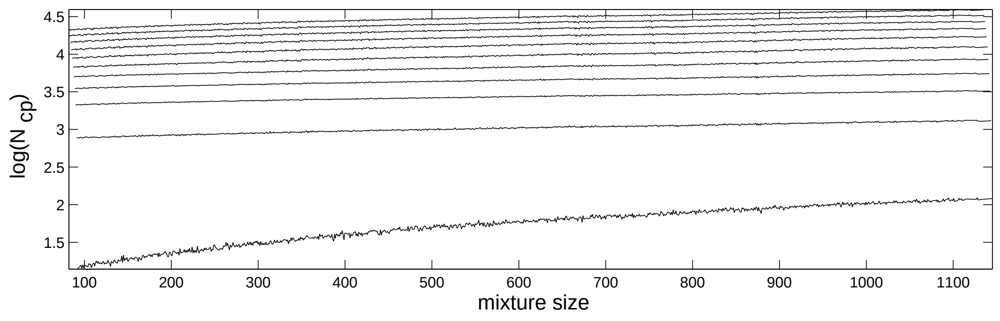
Figure 3.16: Number of candidate proteins for $m = 500$ (left) and $m = 1100$ (right) vs. number of proteins in mixture for $0 \leq f \leq 10$. The error-free case is distinguished by triangular markers.

For a given number of proteins, say $p = 10$, the number of candidate proteins increases with a big factor by introduction of first missing mass, however this trend slows down by increasing f. Figures 3.17a-b show this trend in linear and logarithmic scales. This number similar to error-free case shows a linear trend with respect to mixture size for a fixed p and f

$$N_{cp} = O(m)$$



(a)



(b)

Figure 3.17: Number of candidate proteins versus mixture size for mixtures created by 10 proteins. in (a) linear (b) logarithmic scale. In (a) the error-free value is depicted dotted.

But to see how it behaves by increasing f we fix m and p and observe Ncp. Figure illustrates it for several values of m. There is power-law relationship where $N_{cp} = a(m)f^{b(m)} + c(m)$. The value of $b(m)$ is limited to the interval $[1.62, 1.78]$. This shows that a

$$N_{cp} = O(f^{1.8})$$

or,

$$N_{cp} = O(mf^{1.8})$$

3.4.3 Time complexity

Creating CP

Time complexity in this case also linearly depends on N_{cp} so

$$T_{cp} = O(mf^{1.8})$$

Creating HE

Time for creating list of hyperedges also follows a similar behaviour to error-free case but the amount is larger by a constant value so

$$T_{he} = O(mp^{-1}) = O(m)$$

Creating search tree

Repeating the same approach we employed for error-free case for $1 \leq f \leq 10$ gives a sequence of upperbounds for the base in exponential regression where it is always upperbounded by $b_f(m) = 5 + f/2$, i.e.

$$T_b = O((5 + \frac{f}{2})^m)$$

Overall time complexity

Overall time complexity is

$$T_{all} = T_{cp} + T_{he} + T_b = O(pm^2) + O(m) + O((5 + \frac{f}{2})^{p-N_{iv}})$$

$$= O(pm^2 + (5 + \frac{f}{2})^{p-N_{iv}})$$

3.5 Spurious masses

The algorithm for spurious masses is tested over mixtures containing 1 to 20 proteins with $1 \leq g \leq 20$. The uniqueness check is not carried out for this error type so no improvement is expected from that.

3.5.1 Number of candidate proteins

Number of candidate proteins is almost the same as error-free case since no unknown masses are allowed and the domain is limited to available masses.

3.5.2 Time complexity

Creating CP

Time for creating CP is exactly like error-free case with very small deviations. Therefore

$$T_{cp} = O(N_{cp}) = O(pm^2) \tag{3.10}$$

Creating HE

Time for creating HE is also independent of g and is exactly equal to error-free case, i.e.

$$T_{he} = O(mp^{-1})$$

Essentially, the values totally overlap. This is illustrated in Figure 3.18 for $p = 15$.

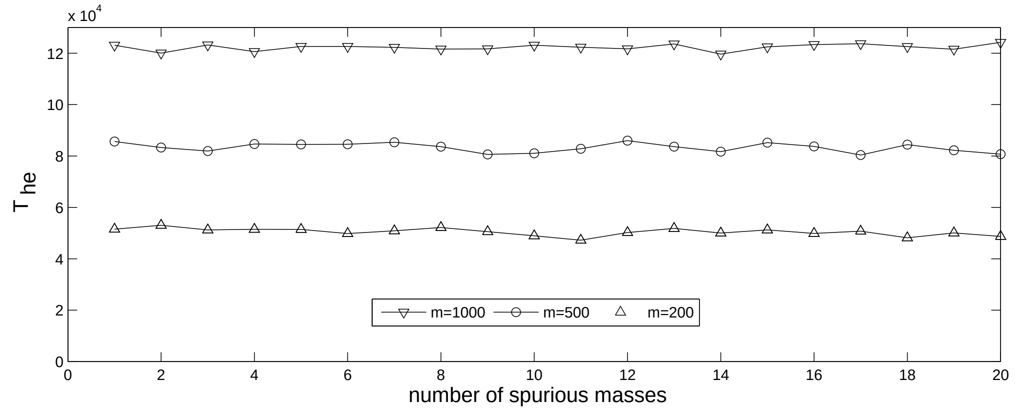


Figure 3.18: Time for creating list of hyperedges is independent of number of spurious masses.

Creating search tree

Repeating the same approach we employed for error-free case for $1 \leq g \leq 10$ gives a sequence of upperbounds for the base in exponential regression where it always decreases by p and is independent of g, and $b_g(m) < 3$ for $p > 5$, i.e.

$$T_b = O(3^m)$$

Overall time complexity

Overall time complexity in case of spurious mass errors is

$$T_{all} = T_{cp} + T_{he} + T_b = O(pm^2) + O(m) + O(3^{p-N_{iv}}) = O(pm^2 + 3^{p-N_{iv}})$$

3.6 Simultaneous errors

The algorithm is implemented in a fashion that is capable of handling both types of error at the same time. A small number of samples, 100 samples for each (f, g) pair where $1 \leq f, g \leq 5$ where created and results where compared to corresponding results in missing and spurious cases and following observations made:

- Similar to spurious masses there is no improvement based on uniqueness check;
- The number of candidate proteins is dictated by f and is essentially independent of g;
- Time for creating HE follows $O(m)$ with insignificant variations by f and g. In fact, these variations are several order of magnitute smaller than average value of HE.
- The branching time however still shows an exponential trend $ab^m + c$ where $\max\{b_f(m), b_g(m)\} < b_{f,g}(m) < \frac{1}{2}(b_f(m) + b_g(m))$, however considering the constant factors, it is always dominated by T_b of missing masses case.

3.7 Future work

There are several ways to improve this approach and create more usable results.

3.7.1 Predicting the number of proteins

The most likely number of proteins in a mixture can be used with a statistical analysis on available database. This can be used as a guideline for setting a relevant K.

3.7.2 Scoring Schemes

Following criteria are suggested for a scoring scheme to rank-order the possible solutions for a given protein mixture:

1. *Close molecular weights*: In most common applications of protein identification, a mixture of proteins with close molecular weights are extracted from polyacrylamide gel bands. This property can be exploited to create a preference for the set of proteins that can be clustered around a single protein.
2. *Consecutive fragments*: One source of spurious fragment masses is the undigested sites that lead to larger fragments. The mass of this larger fragment is the sum of two or more smaller fragments that appear consequently in original protein database. This property can help to give more weight to those proteins.
3. *Origin of proteins*: Higher score can be assigned to solutions that solely contain proteins from one or certain number of specific species or families.

3.7.3 Incomplete digestions

In case of incomplete digestions several extra masses can be reported where each mass is a sum of two or more consecutive fragment masses in a protein. This can be used to determine candidate spurious masses more effectively. This also can be incorporated in scoring scheme for results.

Appendix A

Manual for the accompanying code

The package consists of following routines:

1. `sorter`
2. `unique`
3. `mpconvert`
4. `randommixture`
5. `randompool`
6. `alter`
7. `minimalsets`

All files are treated as text files. For database files (protein and mass files), each line should consist of a set of numbers separated by space(s) where the first number is the ID number (protein ID number or the fragment mass size), and the rest is the corresponding data (list of fragments sets produced by a protein or the set of proteins that produce a given fragment set).

A.1 `sorter`

Usage: `sorter <input file> <output file>`

For each line it preserves the first element (the ID) and for the remaining elements it sorts them and removes duplicate elements.

A.2 `unique`

Usage: `unique <input file> <output file> [xref file]`

This routine is developed to handle the proteins with identical fingerprints in the initial database (see 2.4.2). The code requires two files (input and output database files) and one optional cross reference file. In output file all redundant occurrences of a fingerprint are removed and simply the first protein is kept as a representative. In the cross-reference file a line is then added that starts with the number of representing protein followed by the numbers of removed proteins. For example:

```
Input file:         Output File:        Cross-reference File:
1 10 20 30          1 10 20 30          1 5 6
2 10 30 100         2 10 30 100         4 7
3 40 30 10 97 6     3 40 30 10 97 6
4 100 18 30         4 100 18 30
5 10 20 30
6 10 20 30
7 100 18 30
```

Note that this routine does not sort the elements and it takes into account multiplicity, so the sequence 20, 10, 30, 10 is not equivalent to 10, 20, 30.

A.3 `mpconvert`

Usage: `mpconvert <input file> <output file>`

This routine transforms the table contained in an input file to its cross-reference table. Similar format is presumed for both files where each line begins with an ID followed by the items associated with that ID. For example given that each line in input file contains the protein ID followed by the fragment masses produced by its digestion, the output file will contain the masses as ID and the corresponding protein ID's as items.

A typical example of input and output files are presented as follows:

```
Input file:                             Output File:
1 10 20 30                              6 3
2 30 100 10                             10 1 2 3
3 40 30 10 97 6                         18 4
4 100 18 30                             20 1
                                        30 1 2 3 4
                                        40 3
                                        97 3
                                        100 2 4
```

Note that the items in output file are sorted in an ascending order. The transform is almost self-inverse and applying the routine on output file gives the input file with the exception of order of items. If the items are initially sorted in the input file the result will be identical to initial input file.

A.4 randommixture

Usage:
```
randommixture <No. of proteins:begin> <No. of proteins:end>
              <No. of mixtures> <missing(-)/spurious(+) mass no.>
              <protein file> <output mass file>
```

This routine provides the ideal fragment masses resulted from digestion of a random mixture of proteins. The code randomly chooses a set of proteins (the number of proteins are varied over an interval specified by first two input arguments, and number of mixtures created for each number of proteins is specified by third argument), retrieves the corresponding fragment masses from the proteins data file and outputs the set of masses. The fourth argument is used to introduce a given number of missing (if the argument is preceded by '-') or spurious masses (if the parameter is preceded by a '+'). Simultaneous occurrence of two sources of error are not implemented in this routine. A separate routine `alter` is used to generate the errors from an error-free mixture pool. The masses are sorted and are unique, i.e. the multiplicity of fragment masses are ignored. To increase the readability of the output file some comment lines are added beginning with a hash mark (#). A random mixture of 5 proteins can result in

```
#Protein IDs: 13057 18662 99387 145381 169321
#Fraction masses:
1282 1562 1853 1993 2002 2133 2153 2273 2293 2414 2423 2563 2573
2593 2594 2694 2853 2933 3034 3144 3264 3485 3565 3705 3714 3785
3815 3854 3885 3985 4024 4124 4134 4275 4496 4546 4686 4756 4837
4846 5187 5196 5266 5336 5397 5515 5866 5888 5977 6067 6138 6207
6309 6358 6657 6689 6738 7319 7380 7649 7738 7981 7989 7999 8050
8489 8589 8620 8621 8649 8690 8980 9060 9130 9392 9653 9692 9853
10065 10142 11013 11535 11584 11605 11935 11996 13268 13908 13925
14379 14788 14826 15407 15860 16449 16869 17060 17860 20761 20866
21945 21961 22047 22345 23342 23525 24319 24608 26159 27623 27895
30043 31014 31524 31935 32089 32804 34481 36777 45245 46330 48472
49800 51388 73409 90656
```

Note that all masses are listed on a single line with no line-breaks. That is, in previous example the output file contains three lines.

A.5 randompool

Usage:
```
randompool <No. of proteins:begin> <No. of proteins:end>
           <max group size> <No. of produced mixtures>
           <missing(-)/spurious(+) mass no.> <protein file>
           <output mass file> <distribution file>
```

This routine is developed to produce a big initial pool of random mixtures. It is different from `randommixture` in the sense that it tracks the length of produced mixtures (without multiplicity). It varies the number of proteins over an interval and for each case creates a set of mixtures specified by fourth argument. However, for a given mixture size it only keeps a number of mixtures specified by third parameter. For the pool used in this study following parameters where used

```
randompool 1 50 10 10000000 ...
```

i.e., 10 million mixtures were created for a given number of proteins (varied between 1 and 50) and for each mixture size only 10 first mixtures are written to the output file. However, the distribution of mixture sizes are saved in a separate file, where each row consists of three elements: mixture size, number of mixtures with that size, number of mixtures of that size kept for the pool.

A.6 `alter`

Usage:
```
alter <missing(-)/spurious(+) mass no.> <input mixture file>
      <output mixture file> <masses file>
```

This routine simply removes (or adds) a given number of masses from mixtures in input file. The sign preceding the first argument indicates the type of error, i.e., '+' for spurious masses and '-' for missing masses.

A.7 `minimalsets`

Usage:
```
minimalsets <-k maximum cardinality>
            [-f maximum number of missing masses (0)]
            [-g maximum number of spurious masses (0)]
            [-x mixture data file (mixture)]
            [-c candidate proteins file]
            [-m fragment masses file (masses)]
            [-p proteins file (proteins)]
            [-o output file (minimals)]
            [-v show results on screen]
```

`minimalsets` is the main routine in this thesis[1]. It contains all the routines implemented for creation and manipulation of BST and enumeration of minimal sets using different branching strategies and/or errors in fragments masses. The candidate protein routine is also incorporated in this routine[2]. This makes it possible to look for transversals using an arbitrary set of candidate proteins or the set previously produced by `candidates`. This feature is controlled by `-c` option. The maximum cardinality of transversals must be specified by `-k` option. This is the only obligatory option for the routine since a default value is set for other options.

[1]Note that in the parameters the default values for optional parameters are specified in parenthesis

[2]In the initial edition another routine called `candidates` was also developed which was solely responsible for preprocessing and listing a set of candidate proteins. This was later merged with `minimalsets` routine and further developed. The corresponding command in `minimalsets`is kept for legacy.

To introduce missing/spurious masses, the maximum number of such fragment masses is passed on to the program by `-f` and `-g` options, respectively. Default values are set for protein file, `proteins`, and masses file, `masses` to make it easier for user when one protein database will be used for several runs.

User can override them using `-m` for masses files and `-p` for proteins files. Mixture and output files can be specified by `-x` and `-o` options, respectively. In absence of `-o` a default file name, `minimals`, will be used.

A.8 A typical run

A.8.1 Preparing masses database and finding consistent proteins

Consider that the protein ID's and corresponding fragment masses are contained in a file named `proteins` and we are to create a random mixture of 4 proteins. A typical run can look like this

```
$ ./mpconvert proteins masses
$ ./randommixture 4 proteins mixture
$ ./candidates mixture masses candprots
```

Where the result can be something like this:

```
$ ./mpconvert proteins masses
Reading input file...
Sorting....
Writing output file ....
Completed.
$ ./randommixture 4 proteins mixture
Reading..
192433
Generating random protein ID numbers:
 43978 64481 117544 143737
Creating the list of masses in the mixture: 1282 1562 1853 1882
2133 2153 2293 2313 2414 2423 2433 2563 2573 2594 2694 2703 2853
2914 2933 3004 3034 3104 3144 3184 3194 3264 3285 3305 3364 3405
3424 3485 3564 3695 3696 3714 3724 3745 3785 3836 3885 3984 3985
4035 4136 4185 4276 4326 4336 4406 4435 4436 4555 4557 4576 4678
4705 4707 4726 4745 4837 4856 4936 4986 4996 5026 5107 5137 5286
5347 5405 5417 5447 5477 5507 5526 5529 5566 5696 5707 5787 5819
5837 5847 5918 5977 5979 5997 6027 6097 6149 6268 6276 6287 6310
6319 6329 6418 6477 6488 6538 6548 6638 6639 6728 6758 6809 6849
6969 6989 7018 7069 7079 7147 7158 7220 7441 7451 7521 7580 7590
7599 7610 7668 7680 8030 8059 8091 8169 8189 8262 8330 8370 8531
8641 8681 9000 9013 9041 9061 9179 9269 9352 9501 9510 9641 9730
9792 9883 10071 10092 10154 10190 10202 10283 10301 10373 10411
10433 10494 10543 10592 10594 11325 11585 11817 11846 12055 12085
```

```
12134 12136 12207 12265 12777 12875 13025 13098 13354 13719 13824
14346 14378 14759 14938 14989 15497 15548 15861 15978 16381 16450
16599 16639 17160 17211 17291 17300 17511 17830 17881 18193 18932
18950 19644 19684 19733 20074 20195 20622 20998 22337 22718 22895
23820 24047 26792 27463 27534 28073 29598 30073 30664 34639 34902
52395
```

$ ./candidates mixture masses candprots

```
Mixture contains 225 fragment masses.
Find all proteins that have at least one mass in common with the
mixture...
185155 possible proteins found.
192408 inconsistent proteins found.
Cross out the proteins that result in fragment masses which are
not in the mixture...
There are 25 consistent candidate proteins:
3034 23212 32256 41008 42356 43978 46156 49526 57961 63030 64481
64483 64485 64904 65184 65185 65186 99543 117544 138275 143737
145752 154072 155309 168585
```

A.8.2 Finding transversals

$ ./minimalsets -k 6 -c candprot

```
There are 25 consistent candidate proteins:
3034 23212 32256 41008 42356 43978 46156 49526 57961 63030 64481
64483 64485 64904 65184 65185 65186 99543 117544 138275 143737
145752 154072 155309 168585
There are 225 fragment masses in the mixture.
Creating hyperedges based on fragment masses....
Removing equivalent hyperedges leaves 28 hyperedges:
1: 43978
2: 117544
3: 143737
4: 3034 43978
5: 23212 117544
6: 42356 43978
7: 43978 49526
8: 43978 64904
9: 43978 99543
10: 43978 117544
11: 43978 138275
12: 43978 143737
13: 63030 117544
14: 64904 143737
15: 117544 145752
16: 117544 154072
```

```
17: 117544 155309
18: 64481 64483 64485
19: 41008 64481 64483 64485
20: 43978 57961 64904 143737
21: 43978 64481 64483 64485
22: 63030 64481 64483 64485
23: 64481 64483 64485 117544
24: 64481 64483 64485 168585
25: 32256 46156 64481 64483 64485
26: 64481 64483 64485 65184 65185 65186
27: 3034 43978 64481 64483 64485 117544 143737
28: 43978 64481 64483 64485 64904 117544 143737
Constructing the Bounded Search Tree....
Sorting the results...
All possible solutions:
1: 43978 64481 64483 64485 117544 143737
All minimal solutions:
1: 43978 64481 64483 64485 117544 143737
```

The routine returns a minimal set of size six while originally there were 4 proteins in mixture. This is because proteins 64481, 64483 and 64485 share exact same fragment mass spectrum.

Appendix B

Source code of routines mentioned in the text

B.1 `iseligible`

```
typedef set <unsigned long> ULset;

bool iseligible(ULset &a, ULset &b, UL F)
{
    ULset::iterator i=a.begin(), alast = a.end();
    ULset::iterator j=b.begin(), blast = b.end();
    bool nlastb=true;

    F++;
    for(; i!=alast && F; i++)
    {
        while( (nlastb=(j!=blast)) && (*i>*j) ) j++;
        if(!nlastb) break;
        if(*i<*j) F--;
    }
    for (; i!=alast && F; i++) F--;
    return (0<F);
}
```