2019

ZERO-KNOWLEDGE DE NOVO ALGORITHMS FOR ANALYZING SMALL MOLECULES USING MASS SPECTROMETRY

Patrick Anthony Kreitzberg
University of Montana

ZERO-KNOWLEDGE *DE NOVO* ALGORITHMS FOR ANALYZING SMALL MOLECULES USING MASS SPECTROMETRY

By

Patrick A. Kreitzberg

Bachelor of Science, Oregon State University, Corvallis, OR, 2015

Thesis

presented in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

Spring 2019

Approved by:

Scott Whittenburg Ph.D., Dean
Graduate School

Oliver Serang Ph.D., Chair
Computer Science

Rob Smith Ph.D.
Computer Science

J. Stephen Lodmell Ph.D.
Division of Biological Sciences

Kreitzberg, Patrick A., M.S., June 2019 Computer Science

Zero-knowledge *de novo* algorithms for analyzing small molecules using mass spectrometry

Chairperson: Oliver Serang

In the analysis of mass spectra, if a superset of the molecules thought to be in a sample is known *a priori*, then there are well established techniques for the identification of the molecules such as database search and spectral libraries.

Linear molecules are chains of subunits. For example, a peptide is a linear molecule with an "alphabet" of 20 possible amino acid subunits. A peptide of length six will have $20^6 = 64,000,000$ different possible outcomes. Small molecules, such as sugars and metabolites, are not constrained to linear structures and may branch. These molecules are encoded as undirected graphs rather than simply linear chains. An undirected graph with six subunits (each of which have 20 possible outcomes) will have $20^6 \cdot 2^{\binom{6}{2}} = 2,097,152,000,000$ possible outcomes. The vast amount of complex graphs which small molecules can form can render databases and spectral libraries impossibly large to use or incomplete as many metabolites may still be unidentified.

In the absence of a usable database or spectral library, an the alphabet of subunits may be used to connect peaks in the fragmentation spectra; each connection represents a neutral loss of an alphabet mass. This technique is called "*de novo* sequencing" and relies on the alphabet being known in advance.

Often the alphabet of m/z difference values allowed by *de novo* analysis is not known or is incomplete. A method is proposed that, given fragmentation mass spectra, identifies an alphabet of m/z differences that can build large connected graphs from many intense peaks in each spectrum from a collection.

Once an alphabet is obtained, it is informative to find common substructures among the peaks connected by the alphabet. This is the same as finding the largest isomorphic subgraphs on the *de novo* graphs from all pairs of fragmentation spectra. This maximal subgraph isomorphism problem is a generalization of the subgraph isomorphism problem, which asks whether a graph G_1 has a subgraph isomorphic to a graph G_2. Subgraph isomorphism is NP-complete.

A novel method of efficiently finding common substructures among the subspectra induced by the alphabet is proposed. This method is then combined with a novel form of hashing, eschewing evaluation of all pairs of fragmentation spectra. These methods are generalized to Euclidean graphs embedded in $\mathbb{Z}^n$.

ACKNOWLEDGMENTS

I would first like to thank my ever supporting family. My parents Dawn and Michael Kreitzberg, twin brother Paul Kreitzberg, big brother Daniel Kreitzberg, big sister Tiff Kreitzberg, and biggest brother Jason Harra. As well as Amber Kreitzberg, Dan's wife, and their wonderful son Elliott, Jessica Harra, Jason's wife, and their wonderful children Henry, Oliver, and Vivian. My loving maternal grandparents Roy and Gayle Nelson and paternal grandparents George and Laura Kreitzberg and all cousins, aunts and uncles.

My good friends Franky, Robby, Erik, Aaron, Billy, Tom, and many others.

Thank you to my adviser and mentor, Oliver Serang and to the Serang lab members Kyle Lucke, Max Thibeau, Jake Pennington, and Sean Rice.

Most importantly, the one who always cheers me up when I need it, Butters the goldendoodle.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

Mass spectrometry (MS) is a method for finding the mass of analytes in a sample with the goal of characterizing the sample. These masses are often much too small to use a conventional scale. A sample, which may first be fragmented into its molecular pieces, is ionized. These ions are then separated based on their mass-to-charge ratio (m/z); thus, mass spectrometry actually finds the m/z of the ions, not simply the mass. There are multiple techniques to separate the ions by their m/z values. One technique is time-of-flight (TOF) mass spectrometry. In a TOF machine an ion's m/z is calculated by sending it through a tube with an electric field across it and calculating its velocity (figure 1.1). Tandem mass spectrometry (also known as MS/MS) is a technique in which multiple rounds of mass spectrometry may be performed on a sample. An earlier round of mass spectrometry can be used to select intact molecules by "precursor" mass-to-charge, and then the analytes are fragmented into the molecular components. The mass-to-charge of these fragments are estimated using a subsequent round of mass spectrometry.

The data produced by mass spectrometry is a collection of spectra where each spectrum is typically represented as a series of peaks with each peak representing an ion. A peak's placement along the m/z axis marks the ion's m/z value and the peak's height is the intensity of the ion (figure 1.2). If the molecules in the sample fragment when performing mass spectrometry, the output is typically referred to as

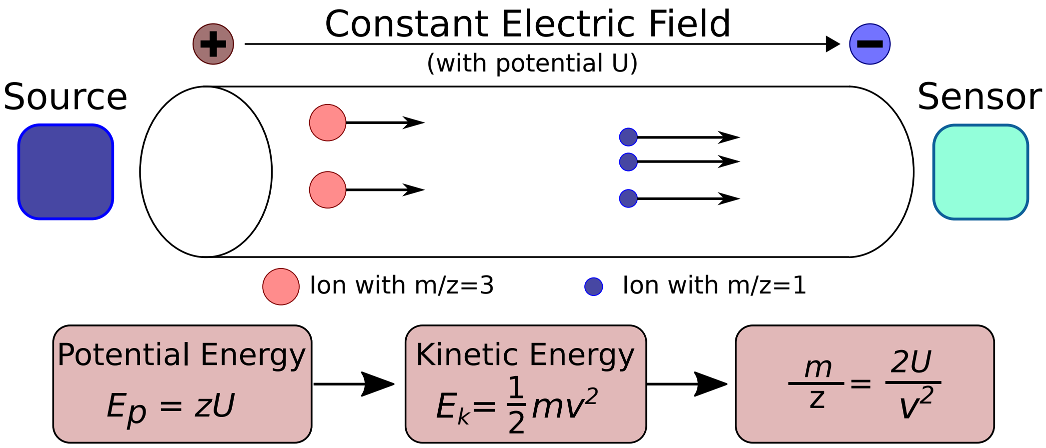


Figure 1.1 **Diagram of time-of-flight mass spectrometry.** An ionized sample is put into a tube which has a constant electric field across it. The electric field causes the ionized sample to move through the tube. From the constant electric field, the potential energy, which is converted into kinetic energy, can be calculated. From the potential and kinetic energy, the velocity, which depends on the m/z of an ion, may be calculated. If two ions have the same charge, the lighter one will reach a higher speed.

fragmentation spectra. Intensity values can be thought of as roughly proportional to the abundance of the ion, so a single peak does not represent a single ion in the sample, it may represent many ions with the same m/z value. The scale of the intensity values are not necessarily the same for all mass spectrometry techniques or all machines.

The difference between two peaks m/z value is also an m/z value. Sometimes, but not always, this m/z value represents another ion in the sample, or one that used to be in the sample such as a metabolite which was created then destroyed during digestion. For example, if the difference between two peaks is the mass of water, then it is possible the larger peak lost a water molecule with charge 1 to become the smaller peak.

The mass spectrometric analysis of structured molecules is important for analysis of glycoconjugates [1] and for drug discovery [2]. Often these methods cannot rely on

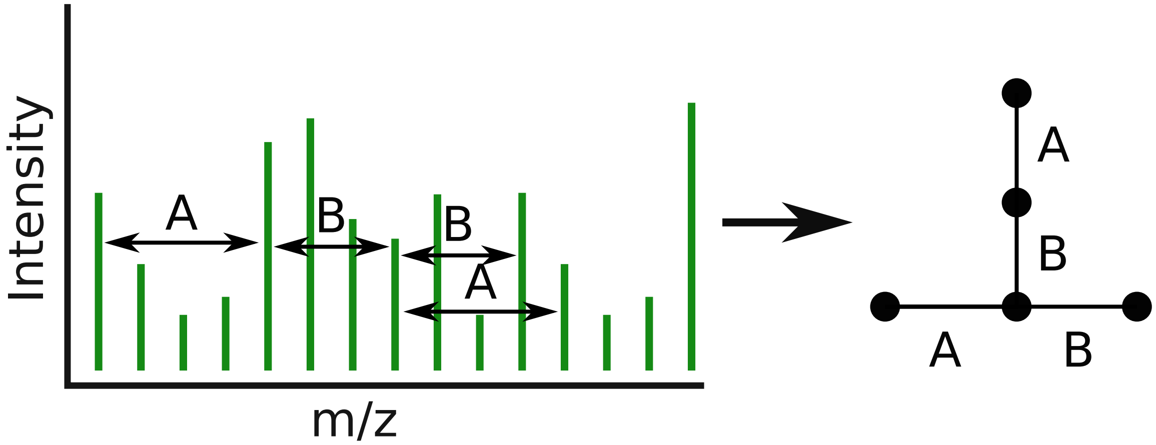


Figure 1.2 **On the left, a representation of a spectrum with a set of peaks that are linked together by *de novo* sequencing. On the right, a graph with a bijection to the set of peaks linked together by *de novo* sequencing** A cartoon of a typical representation of data generated by mass spectrometry with peaks linked together by masses in the alphabet {A, B}. Each peak represents an ion whose m/z value is the same as the peak. The height of the peak is the intensity of the ion, which can be thought of as roughly proportional to the abundance of the ion. Peaks are linked by a mass in the alphabet if the difference in the peaks is equal to the mass divided by some charge. The graph is created by forming a bijection between the nodes in the graph and the set of peaks in the spectrum connected by *de novo* sequencing. An edge exists between two nodes if the two peaks they are projected onto are connected by an m/z value in the alphabet.

machine-generated databases (as can often be done for peptide search) because of the combinatoric nature of these small molecules, which would make a machine-generated database far too large to use. Fragmentation trees may be used for analysis of small molecules where databases may not exist or are too large, but they rely on enumerating all molecular formulas that match the precursor mass [3]. Enumerating over all molecular formulas for a precursor mass can become very costly, particularly for a larger precursor mass or with a fairly imprecise mass-to-charge measurement; thus, fragmentation trees may not be suitable in all cases. Spectral libraries generated by known small molecule content can be used, but they need to be painstakingly curated from a myriad of potential molecules of interest; therefore, even if the resources are available to do so, they may not be suitable for applications that include unexpected compounds or reactions. Likewise, when an MS1 spectrum is generated by few intact molecules, it may be possible to isolate the most abundant mass in the spectrum using only Fourier analysis [4].

To date, *de novo* approaches, which link peaks in fragmentation spectra when they are different by a mass in the alphabet, are the best tools for these problems. For example, *de novo* peptide sequencing may be performed using an alphabet of 20 amino acid masses, whereas *de novo* glycan analysis may be performed using an alphabet of four common sugar residues. Once an alphabet is known, dynamic programming can be used to link peaks for linearly chained molecules (*e.g.*, peptides) [5][6] or arbitrarily structured small molecules (*e.g.*, sugars) [7][8]. The ability to use certain “characters” in the alphabet can also be constrained to an arbitrary flow chart (for instance, it may state that a peptide with more than two of a given amino acid should not be considered) by performing dynamic programming on the Cartesian products between the graph of linked peaks and the flow chart from the constraints [9]. Distinctions between fragmentation spectra can also be used to build graphs for a given alphabet by clustering spectra to find highly similar neighbor spectra and then attempting to match small changes between these neighboring spectra using the given alphabet [10]. Approaches reminiscent of this can be used to better characterize biochemical pathways [11].

All above approaches need to know the alphabet, *i.e.*, the masses considered during the *de novo*; however, in a truly blind *de novo* application, this alphabet will not be known. This is important when identifying active compounds and therapeutic components in venoms [12] or plant products [13] and can similarly be significant when finding drug metabolites produced. Even fundamental chemical components of the sugar alphabet (such as O-GlcNAc-P) were only discovered relatively recently [14]; thus, if there are more undiscovered components of the sugar alphabet, then any current sugar alphabet will be incomplete and a blind approach may be the only way to use these undiscovered sugars in an alphabet.

Two approaches with partially blind aspects to them are the offset frequency function and spectral networks. The offset frequency function, introduced by Dančík et al. [15], builds a *de novo* graph using the amino acid alphabet, and then builds the empirical distribution of peak differences between peaks in the *de novo* peptide path and peaks not in the *de novo* peptide path; however, this approach needs to know the amino acid alphabet in advance. Spectral networks [16] are likewise used for the analysis of peptides. For example, a pair of spectra matching peptides with either overlapping sequences (*e.g.*, `EEAMPN` and `AMPNGGR`)– or a pair of modified and unmodified peptide spectra– can be matched by sequence overlap after database search and then differing peaks in a spectral pair can elucidate sequence changes, modifications, *etc.* Like the offset frequency function, this approach relies on knowledge of the amino acid alphabet and methods for sequencing peptide spectra (either *de novo* or database search) via that amino acid alphabet.

In this paper we introduce an approach to perform blind *de novo* analysis of mass spectra, and to estimate an alphabet from a collection of spectra (*i.e.*, the “alphabet projection of the spectra”). Our approach seeks to find the alphabet that would best explain the most high-intensity peaks and simultaneously build the largest connected graphs. This approach is also informative as to which peaks can be linked by this alphabet; the graph produced by linking peaks in a *de novo* manner can be helpful to inferring the chemical structure of a compound. In this manner, the method proposed can also be seen as an unsupervised *de novo* approach (*i.e.*, a *de novo* approach where the alphabet is not known in advance).

When *de novo* sequencing is applied to spectra it is possible some real structure is connected to an extraneous peak by random chance. It is unlikely this same peak is connected by random chance in multiple spectra, but the connected peaks in the underlying structure should almost always be there if the compound is. This recurring structure is trustworthy; if found throughout multiple spectra, it is likely to be important to the sample. Finding different compounds with similar chemical structures may also play an important role in pharmacological research. For example, the drugs Famprofazone and Deprenyl both metabolize into amphetamine and methamphetamine which have very similar structures [17][18].

A new approach is introduced to efficiently find recurring chemical structures in many spectra. The approach develops a new method for performing maximal subgraph isomorphism on *de novo* graphs from two spectra. It is then improved with locality sensitive hashing to do this without comparing all pairs of fragmentation spectra. This method is generalized to efficiently find isomorphic subgraphs among graphs embedded in $\mathbb{Z}^n$.

CHAPTER 2 METHODS

Variable	Meaning
$s^{(\ell)}$	The indices of peaks in a fragmentation spectrum ℓ.
$m_i^{(\ell)}$ for $i \in s^{(\ell)}$	The m/z of peak i in spectrum $s^{(\ell)}$.
$p_i^{(\ell)}$ for $i \in s^{(\ell)}$	The intensity of peak i in spectrum $s^{(\ell)}$.
$D^{(\ell)}$	The set of $s^{(\ell)}$, $m^{(\ell)}$, and $p^{(\ell)}$ for spectrum ℓ.
$m_j^{(\ell)} - m_i^{(\ell)}$ for $i \in s^{(\ell)}, j \in s^{(\ell)}$	The m/z difference between peaks j and i in spectrum $s^{(\ell)}$.
$\Delta_1, \Delta_2, \Delta_3, \ldots \Delta_d$	The alphabet of size d (units are mass, not m/z).
ϵ	The maximum allowed error tolerance in m/z; if two m/z values are approximately equal, the absolute value of their difference must be $\leq \epsilon$.
$E_{z,i,j,k}^{(\ell)}$ for $i \in s^{(\ell)}, j \in s^{(\ell)}, k \in \{1, 2, \ldots d\}$	1 if peaks i and j in spectrum $s^{(\ell)}$ can be connected by difference Δ_k using charge z; *i.e.*, $\lvert m_j^{(\ell)} - m_i^{(\ell)} - \frac{\Delta_k}{z} \rvert \leq \epsilon$ for some charge state. 0 otherwise.
$E_z^{(\ell)}$	The set of all edges for spectrum ℓ that use charge state z.
$g(E_z^{(\ell)}) = \{e_1, e_2, \ldots\}$	The collection of edges in the connected components of the graph defined by $E_z^{(\ell)}$.
θ	A hyper-parameter that can be tuned to influence the acceptance rate. $\theta = 0$ will accept all proposed changes, $\theta = \infty$ will only accept changes that improve the likelihood.

Table 2.1 **This table defines the notation used throughout the paper.** In each spectrum $s^{(\ell)}$, a peak $i \in s^{(\ell)}$ has m/z value $m_i^{(\ell)}$, with machine tolerance ϵ, and intensity $p_i^{(\ell)}$; the set of $s^{(\ell)}$, $m_i^{(\ell)}$, and $p_i^{(\ell)}$ for all peaks in spectrum ℓ form $D^{(\ell)}$. The alphabet of size d, $\Delta_1, \Delta_2, \Delta_3, \ldots \Delta_d$, is used to form a set of edges, $E_z^{(\ell)}$, for charge state z and the set of edges form connected components, $g(E_z^{(\ell)}) = \{e_1, e_2, \ldots\}$, of the graph defined by $E_z^{(\ell)}$.

We use the notation from Table 2.1 to formalize the alphabet projection problem:

We use variables i and j to index peaks in the spectra, while we use variable k to index the alphabet. For variables i and j, assume that the indices are ordered so that the masses are sorted in ascending order: $m_j^{(\ell)} > m_i^{(\ell)} \leftrightarrow j > i$.

2.1 Convex optimization

2.1.1 Linear and quadratic programming

Linear programming is a method to solve a class of optimization problems characterized by an objective function and a set of constraints. For instance, if you are a diary farmer, optimization can help you decide what amount of butter, 2% milk, skim milk, etc. to create to maximize your profits. The constraints would be the total amount of milk you have from your cows, how much milk-fat it takes to make each product, etc. The solution will be a set of variables: volume of butter to make, 2% milk, skim milk, etc. and the optimal value of the objective function. Linear programs (LPs) with linear constraints can be solved efficiently, even in the worst case [19].

Formally, in linear programming the aim is to optimize an linear objective function according to a set of constraints which are affine inequalities. An affine equation is a linear equation plus a constant. The problems are usually represented using matrix notation:

$$\begin{aligned}&\text{minimize}\\&\qquad c^T \cdot x\\&\text{such that}\\&\qquad A \cdot x \leq b\\&\qquad x \geq 0.\end{aligned}$$

An integer linear program (ILP) is an LP in which all the variables are confined to be integers. Minimum vertex cover can be reduced to an ILP, proving that solving ILPs is NP-hard. For a graph $G(e, v)$, the minimum vertex cover problem finds a set of nodes in G such that every edge in G is connected to a node in the set. The following ILP can solve minimum vertex cover:

$$\text{minimize}$$

$$\sum_{v \in V} y_v \tag{2.1a}$$

$$\text{subject to}$$

$$\forall uv \in E,\ y_v + y_u \geq 1 \tag{2.1b}$$

$$\forall v \in V,\ y_v \geq 0 \tag{2.1c}$$

$$\forall v \in V,\ y_v \in \mathbb{Z} \tag{2.1d}$$

In ILP (2.1), y_v is an indicator variable which is which is at least one if vertex v is included in the set cover. Constraint (2.1b) forces at least one vertex for every edge to be in the set cover. Constraints (2.1c) and (2.1d) force y_v to be either zero or at least one. Then minimizing over the sum of the indicator variables is the same as finding the minimum set cover.

Mixed integer programs (MILP) are linear programs where some variables are continuous and some are confined to be integers. MILPs are also NP-hard as they are a generalized version of ILPs.

A quadratic program (QP) is similar to an LP but the objective function has quadratic terms. In general, solving QPs is NP-hard [20]; though, there are special cases where they can be solved efficiently.

If an LP, ILP, MILP, or QP has quadratic constraints then the program is "quadratically constrained" and their abbreviations are prefixed with "QC". Making more general constraints is typically considered more difficult than doing so to the objective function.

2.1.2 Original quadratically constrained linear program

The first approach towards finding an alphabet was to use convex optimization to minimize the size of the alphabet while maximizing the number of high intensity peaks connected by the alphabet. This was the very first formulation of a QCLP:

minimize

$$\|\Delta\|_2 \tag{2.2a}$$

subject to

$$\sum_i p_i q_i \geq \gamma \tag{2.2b}$$

$$\forall i,\ q_i \leq \sum_{j,k} e^{(\ell)}_{i,j,k} \tag{2.2c}$$

$$\forall i,\ q_i \in \{0, 1\} \tag{2.2d}$$

$$\forall i, j, k,\ e^{(\ell)}_{i,j,k} \in \{0, 1\} \tag{2.2e}$$

$$\forall k,\ \forall i \neq j \in s,\ \ -\epsilon \leq e^{(\ell)}_{i,j,k} (m_j - m_i - \Delta_k) \leq \epsilon \tag{2.2f}$$

$$\forall k \in \{1, 2, \ldots n\},\ \Delta_k \in \{1 \times N, 2 \times N, \ldots, d \times N\}. \tag{2.2g}$$

The inputs s, n, γ, ϵ are all seen as constant by the linear program. q_i values are binary indicator variables which are 1 if peak i is touched by a Δ in the alphabet. Constraint (2.2b) requires the sum of the intensities of peaks connected by the alphabet to be at least γ, which is a hyper-parameter. Constraint (2.2f) forces the Δ_k values to be near $m_j - m_i$ for some $i, j \in s$ and then constraint (2.2f) turns on $e^{(\ell)}_{i,j,k}$ if m_i and m_j can be connected by Δ_k.

These constraints set up the bounds of the problem, then the linear program will minimize the alphabet in order to minimize the objective function.

Constraint (2.2g) was never implemented but could further refine the alphabet by only allowing values which were multiples of a neutron mass; however, this would not always be realistic, for there are plenty of compounds which are not an integer multiple of the neutron mass (e.g. the atomic mass of alanine is 70.4... × N).

2.1.3 Minimal quadratically constrained linear program

In order to reduce the number of variables of (2.2), the QCLP was compressed to an equivalent problem that excludes the q_i variables. Constraint (2.3d) has a quadratic term, $e^{(\ell)}_{i,j,k} \cdot \Delta_k$, so the problem is still QC.

minimize

$$\|\Delta\|_2 \tag{2.3a}$$

subject to

$$\sum_i p_i \cdot \left(\sum_{j \neq i,k} (e^{(\ell)}_{i,j,k} + e^{(\ell)}_{j,i,k}) \right) \geq 1 \tag{2.3b}$$

$$\forall i, j, k, \;\; e^{(\ell)}_{i,j,k} \in \{0, 1\} \tag{2.3c}$$

$$\epsilon \geq e^{(\ell)}_{i,j,k}(m_j - m_i - \Delta_k) \geq -\epsilon. \tag{2.3d}$$

2.1.4 Lagrangian relaxation

Lagrangian relaxation is a method of approximating a problem with a difficult constraint by using a series of problems with simpler constraints and more complex objective functions. The new problem should be many times faster to solve so that in between iterations the parameters can be adjusted. In this case we use Lagrangian relaxation to move the quadratic constraints $\epsilon \geq e^{(\ell)}_{i,j,k}(m_j - m_i - \Delta_k) \geq -\epsilon$ into the objective function. Each new term in the objective function is multiplied by a unique Lagrange multiplier $\lambda_{i,j,k}$. In the view of the quadratic program solver the Lagrange multipliers are constants, weighting the influence of the constraints moved into the objective function. The multipliers are iteratively updated outside the QP solver.

$\lambda_{i,j,k}$ are adjusted after each time the simpler problem is solved. The adjustments are made using gradient ascent over the lambda values; therefore, we want to reward constraints which have large lambda values. Since we are minimizing, we want the large lambda values to be multiplied by a negative value so we make the terms in the objective value be negative when the constraint is obeyed.

Constraint	Lagrangian objective function term
$\epsilon \geq e_{i,j,k} \cdot (m_j - m_i - \Delta_k)$	$\lambda_{i,j,k} \cdot (e_{i,j,k} \cdot (m_j - m_i - \Delta_k) - \epsilon)$
$-\epsilon \leq e_{i,j,k} \cdot (m_j - m_i - \Delta_k)$	$\lambda_{i,j,k} \cdot (-e_{i,j,k} \cdot (m_j - m_i - \Delta_k) - \epsilon)$

Unfortunately, this causes a problem because the two Lagrangian terms cancel each other out to sum to a constant. Thus, we have to combine the two constraints into one. This can be done simply by squaring the $(m_j - m_i - \Delta_k)$ term and subtracting ϵ^2 (this did not seem to differ from using the absolute value function).

The downside is now we have a cubic function since we have the term $e_{i,j,k}^{(\ell)} \cdot \Delta_k^2$.

We also want to encourage more peaks to be turned on. A trick in convex optimization is to introduce a term into the objective function, $M \cdot r$, where M, a constant, is large and r, a variable, is between 0 and 1. To optimize the problem, r will want to be lowered in order to shrink $M \cdot r$. We use this to encourage more connectivity by adding the constraint $\sum_i p_i \cdot q_i + M \cdot r \geq \gamma$. This encourages more q_i values to be turned on in order to lower the objective function while holding the constraint. This leads to the following QP:

minimize

$$\|\Delta\|_2 + \sum_{i,j,k} \lambda^2_{i,j,k,0} \cdot e^{(\ell)}_{i,j,k}((m_j - m_i - \Delta_k)^2 - \epsilon) + M \cdot r \tag{2.4a}$$

subject to

$$\sum_i p_i \cdot q_i + M \cdot r \geq \gamma \tag{2.4b}$$

$$\forall i,\ \ q_i \in \{0,1\} \tag{2.4c}$$

$$\forall i, j, k,\ \ e^{(\ell)}_{i,j,k} \in \{0,1\} \tag{2.4d}$$

$$\forall i,\ \ q_i \leq \sum_{i,j,k} (e^{(\ell)}_{i,j,k} + e^{(\ell)}_{j,i,k}) \tag{2.4e}$$

$$0 \leq \Delta_k \leq \max_k(\Delta_k) \tag{2.4f}$$

$$0 \leq r \leq 1. \tag{2.4g}$$

2.1.5 Minimizing over indicator variables

Minimizing over $\|\Delta\|_2$ will keep the alphabet small in size, but it will also select Δ_k values which are small in magnitude; this is something we do not want to favor. In order to minimize the cardinality of the alphabet, and not the values in it, we can minimize over some binary variables b_k which are set to 1 if the corresponding Δ_k is used in the alphabet. Thus (2.4a) becomes

$$\sum_k b_k + \sum_{i,j,k} \lambda^2_{i,j,k,0} \cdot e^{(\ell)}_{i,j,k}((m_j - m_i - b_k \cdot \Delta_k)^2 - \epsilon) + M \cdot r.$$

2.1.6 An edge-centric model: incentivizing larger graphs

In order to promote connectivity, which promotes more non-zero $e^{(\ell)}_{i,j,k}$ terms, we add the following term to the objective function: $C/(\sum_{i,j,k} e^{(\ell)}_{i,j,k})$, for some constant $C \gg 1$. This term will dominate the objective function for large enough C so the priority for minimization will be to increase the denominator.

2.1.7 Battle of solvers: Mathematica versus CPLEX

Up until this point we were using Mathematica to solve the optimization problems. Before Lagrangian relaxation Mathematica would hang on any problem with $d > 16$. Once we switched to CPLEX the runtimes were dramatically better. CPLEX is made by IBM and is a state-of-the-art convex optimization solver.

2.1.8 Removing quadratic constraints

Now using CPLEX, we are able to return to the pre-Lagrangian relaxation version which was an LP with quadratic constraints. The quadratic constraint (2.3d) was made of a binary variable multiplied by a continuous variable. In this case it is possible to relax the QP to an LP [21].

Let b be a binary variable and x a continuous variable where L and U are the lower and upper bounds for x, respectively. The term $b \cdot x$ can be relaxed by being replaced with a new continuous variable, z, and adding the following constraints:

$$z \le U \cdot b$$

$$z \ge Lb$$

$$z \le x - L \cdot (1 - b)$$

$$z \ge x - U \cdot (1 - b).$$

We can implement this trick to replace the quadratic constraint (2.3d) with constraints (2.5c) through (2.5h) by substituting $z_{i,j,k}$ for $e^{(\ell)}_{i,j,k} \cdot \Delta_k$.

2.1.9 Maximize number of edges and finding unique masses through constraints

Here, we maximize over the edge indicator variables to try and improve the amount of edges turned on. Constraint (2.7i) is added which allows only one Δ_k to connect the same peak pair so redundant alphabet masses do not help the objective function. These two constraints combine to enable the LP to find all unique masses.

minimize

$$\sum_{i,j,k} b_k - \gamma \sum_{i,j,k} \cdot e^{(\ell)}_{i,j,k} \tag{2.5a}$$

subject to

$$\forall i, j, k,\ e^{(\ell)}_{i,j,k} \in \{0, 1\} \tag{2.5b}$$

$$\forall i, j, k,\ z_{i,j,k} \leq \epsilon \tag{2.5c}$$

$$\forall i, j, k,\ z_{i,j,k} \geq -\epsilon \tag{2.5d}$$

$$\forall i, j, k,\ z_{i,j,k} \leq e^{(\ell)}_{i,j,k}(m_j - m_i) \tag{2.5e}$$

$$\forall i, j, k,\ z_{i,j,k} \geq e^{(\ell)}_{i,j,k}(m_j - m_i - \max_k(\Delta_k)) \tag{2.5f}$$

$$\forall i, j, k,\ z_{i,j,k} \geq (m_j - m_i - \Delta_k) - (m_j - m_i) \cdot (1 - e^{(\ell)}_{i,j,k}) \tag{2.5g}$$

$$\forall i, j, k,\ z_{i,j,k} \leq (m_j - m_i - \Delta_k) - (m_j - m_i - \max_k(\Delta_k)) \cdot (1 - e^{(\ell)}_{i,j,k}) \tag{2.5h}$$

$$\forall i, j,\ 1 \geq \sum_k e^{(\ell)}_{i,j,k} \tag{2.5i}$$

$$\forall i, j, k,\ b_k \geq e^{(\ell)}_{i,j,k} \tag{2.5j}$$

2.1.10 Two step method of finding the best Δs and then minimizing the alphabet

Inspired by the Lagrangian mentality of solving many smaller problems, we modified our approach to only solve (2.5) for one Δ_k at a time (2.6). This first pass creates an alphabet which is not minimized in size but maximized in number of peaks connected. The second pass (2.7) is used to minimize the alphabet whose values are now all constant but can be minimized in size by excluding some masses.

In the first pass, the b_k indicator variables are now useless, so we modify (2.5a) by removing the first term and constraint (2.7i). After each iteration, k, all constraints have the terms removed for pairs i, j such that $\epsilon \geq m_j - m_i - \Delta_k \geq -\epsilon$ is satisfied. This has the effect of having significantly fewer constraints and variables which decrease even more over time.

Let I_k and J_k be the set of i and j values, respectively, such that $\epsilon > |m_j - m_i - \Delta_h|$ for $h \in \{0, .., k-1\}$ where k is the current iterations of the first pass. Then the LP being optimized in each iteration of the first pass is as follows:

minimize

$$-\gamma \sum_{i,j} \cdot e^{(\ell)}_{i,j,k} \tag{2.6a}$$

subject to

$$\forall i \in I, j \in J \; e^{(\ell)}_{i,j,k} \in \{0, 1\} \tag{2.6b}$$

$$\forall i \in I, j \in J, \; z_{i,j,k} \leq \epsilon \tag{2.6c}$$

$$\forall i \in I, j \in J, \; z_{i,j,k} \geq -\epsilon \tag{2.6d}$$

$$\forall i \in I, j \in J, \; z_{i,j,k} \leq e^{(\ell)}_{i,j,k}(m_j - m_i) \tag{2.6e}$$

$$\forall i \in I, j \in J, \; z_{i,j,k} \geq e^{(\ell)}_{i,j,k}(m_j - m_i - \max_k(\Delta_k)) \tag{2.6f}$$

$$\forall i \in I, j \in J, \; z_{i,j,k} \geq (m_j - m_i - \Delta_k) - (m_j - m_i) \cdot (1 - e^{(\ell)}_{i,j,k}) \tag{2.6g}$$

$$\forall i \in I, j \in J, \; z_{i,j,k} \leq (m_j - m_i - \Delta_k) - (m_j - m_i - \max_k(\Delta_k)) \cdot (1 - e^{(\ell)}_{i,j,k}). \tag{2.6h}$$

After (2.6) has been solved d times to find an alphabet, the following LP is solved to minimize the alphabet:

minimize

$$\sum_k b_k - \gamma \sum_{i,j,k} \cdot e^{(\ell)}_{i,j,k} \tag{2.7a}$$

subject to

$$\forall i,j,k,\ e^{(\ell)}_{i,j,k} \in \{0,1\} \tag{2.7b}$$

$$\forall i,j,k,\ z_{i,j,k} \leq \epsilon \tag{2.7c}$$

$$\forall i,j,k,\ z_{i,j,k} \geq -\epsilon \tag{2.7d}$$

$$\forall i,j,k,\ z_{i,j,k} \leq e^{(\ell)}_{i,j,k}(m_j - m_i) \tag{2.7e}$$

$$\forall i,j,k,\ z_{i,j,k} \geq e^{(\ell)}_{i,j,k}(m_j - m_i - \max_k(\Delta_k)) \tag{2.7f}$$

$$\forall i,j,k,\ z_{i,j,k} \geq (m_j - m_i - \Delta_k) - (m_j - m_i) \cdot (1 - e^{(\ell)}_{i,j,k}) \tag{2.7g}$$

$$\forall i,j,k,\ z_{i,j,k} \leq (m_j - m_i - \Delta_k) - (m_j - m_i - \max_k(\Delta_k)) \cdot (1 - e^{(\ell)}_{i,j,k}) \tag{2.7h}$$

$$\forall i,j,k,\ b_k \geq e^{(\ell)}_{i,j,k}. \tag{2.7i}$$

2.1.11 Maximum vertex cover approach

A bipartite graph is a graph where the nodes can be separated into two classes such that there are no edges between two nodes in the same class. Here, the two classes are the collection of Δ_k values and the collection of edge indicator variables (2.1). The goal of this model is to maximize the coverage of the edge indicator variables class by selecting the best subset of the Δ class.

All values $m_j - m_i$ have been calculated in advance and each Δ_k has been assigned to a unique (within ϵ) $m_j - m_i$ value. In this model, the alphabet values are limited to be below 400Da; this is a way to limit the number of variables and improve the speed.

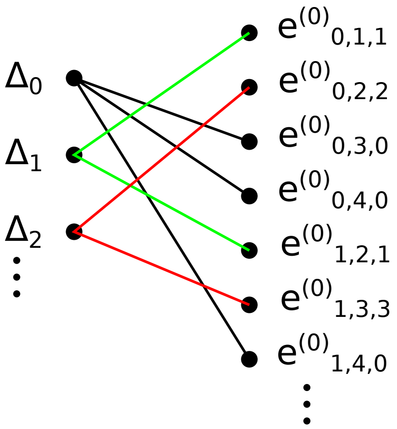


Figure 2.1 Diagram of the bipartite relationship modeled in a linear program (2.8). An edge connects a Δ value to an edge indicator variable $e^{(\ell)}_{i,j,k}$ if $\exists z \in \{1,2,3\}$ $s.t.$ $|m_j - m_i - \Delta_k/z| \leq \epsilon$.

Now, Δ_k in the linear program is a binary variable indicating whether or not Δ_k will be in the alphabet. Constraint (2.8f) limits the number of turned on Δ_k values to d, the size of the alphabet. This means that we do not minimize the alphabet, instead it is set to a fixed size. An edge exists between Δ_k and $e^{(\ell)}_{i,j,k}$ in the bipartite graph if $\exists z \in \{1,2,3\}$ $s.t.$ $|m_j - m_i - \Delta_k/z| \leq \epsilon$. This is the first model to take charge, z, into account. Edge indicator variables which do not have an edge in the bipartite graph are removed ($i.e.$ $e^{(\ell)}_{i,j,k}$ does not exist if $\forall z \in \{1,2,3\}$, $|m_j - m_i - \Delta_k/z| > \epsilon$). Thus, there are many less $e^{(\ell)}_{i,j,k}$ variables but many many more Δ_k variables. This is now an ILP where all variables are binary; this is sometimes called a binary linear program or zero-one linear program.

Here, q_i indicator variables are used again to indicate if peak i is touched by a mass in the alphabet. Constraint (2.8i) forces q_i to be turned on if any edge connecting peak i is on. Δ_k indicator variables are forced on if any edge indicator variable which uses Δ_k is turned on through constraint (2.8g). Δ_k must be forced to turn on if any edge indicator variables which connect peaks using Δ_k are turned on. This can be done by using the constraint $\sum\limits_{i,j,k} e_{i,j,k}^{(\ell)} \leq \Delta_K$; however, this would limit the amount of edges using Δ_k to one. A large constant, M, is used in constraint (2.8g) to allow all edge indicator variables which use Δ_k to be turned on and still satisfy the constraint $\sum\limits_{i,j,k} e_{i,j,k}^{(\ell)} \leq M \cdot \Delta_K$.

In order to promote a connected graph, and not just a series of disconnected edges, the a_i variables are used along with (2.8e) and (2.8h). Constraint (2.8h) says that if any edge touching peak i is turned on, then at least one more edge must be turned on or a_i must be turned on. Then (2.8e) forces at most one a_i to be on, meaning all but one peak must have either zero or strictly more than one edge connecting it. In this way, each peak must be connected to at least one other peak, forcing one big connected graph. Multiple connected graphs may be allowed by increasing the right-hand-side of (2.8e).

minimize

$$-\sum_{i,j} e^{(\ell)}_{i,j} \tag{2.8a}$$

subject to

$$\forall k,\ \Delta_K \in \{0,1\} \tag{2.8b}$$

$$\forall i,j,k,\ e^{(\ell)}_{i,j,k} \in \{0,1\} \tag{2.8c}$$

$$\forall i,\ a_i, q_i \in \{0,1\} \tag{2.8d}$$

$$\sum_i a_i \leq 1 \tag{2.8e}$$

$$\sum_k \Delta_k \leq d \tag{2.8f}$$

$$\sum_{i,j,k} e^{(\ell)}_{i,j,k} \leq M \cdot \Delta_K \tag{2.8g}$$

$$\forall i,j,k,\ell,\ a_{i,\ell} + \sum_{j,\ell}(e^{(\ell)}_{i,j>i,\ell} + e^{(\ell)}_{j<i,i,\ell}) \leq 2q_i \tag{2.8h}$$

$$\forall i,\ell,\ \sum_j (e^{(\ell)}_{i,j>i,\ell} + e^{(\ell)}_{j>i,i,\ell}) \leq M \cdot q_i. \tag{2.8i}$$

2.2 A max-flow/min-cut formulation

Here, we utilize graph-cuts in order to obtain an alphabet. A graph cut is a separation of the nodes of the graph into two disjoint subsets. We form weighted, directed graphs based on Δ_k values, edges, peaks, and peak intensities. The Δ_k values are not variables, but are a constant, calculated from $m^{(\ell)}_j - m^{(\ell)}_i$ for some i, j, and ℓ. Redundant Δ_k values (ones within ϵ of each other) are merged together. We include a source node labeled "NOT USED", and a sink node labeled "USED". Graph cuts we look at will always put the NOT USED and USED nodes into different sets which we label the "not used" and the "used," respectively. We want to find the minimum cut, which is the graph cut that minimizes the sum of the weights of the edges which go from the not used nodes to the used nodes. The used nodes form the alphabet and the minimization of the cut will be what optimizes the alphabet.

The max-flow min-cut theorem states that the maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in the min-cut [22]. An efficient way to find the min-cut is to solve for the max-flow by formulating the max-flow problem as an LP and solving via convex optimization. In contrast, the max-cut problem is NP-hard [23].

If a model has both Δ_k nodes and edge nodes, $e^{(\ell)}_{i,j}$, they are connected with a weight of ∞ if $|m^{(\ell)}_j - m^{(\ell)}_I - \Delta_k| \leq \epsilon$. The weight is infinite because including the Δ_k in the alphabet necessarily forms the edge; so this can never be cut.

2.2.1 Edge and peak centric model

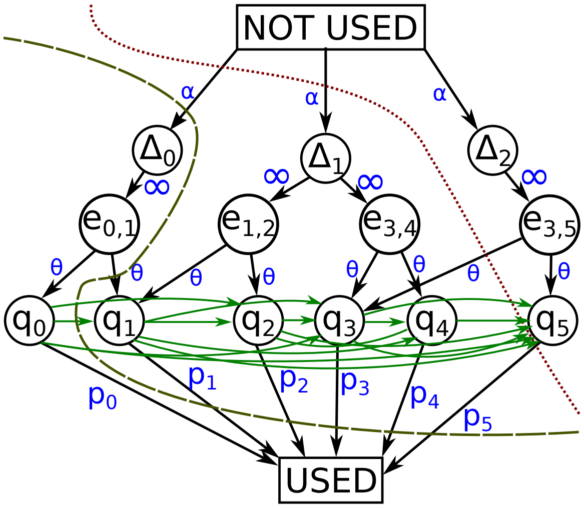


Figure 2.2 **Edge and peak centric graph-cut model** used to find a minimum alphabet. All solid green edges between q_i nodes have a weight of β. The dotted dark red line represents a graph cut which would generate $\Delta = \{\Delta_0, \Delta_1\}$ and the dashed dark green line represents a graph cut which would generate $\Delta = \{\Delta_0\}$. Large connected graphs are encouraged by the use of the β weighted edges.

An edge-and-peak centric model (2.2) has nodes for the Δ_k values, the edges, and the peaks. These peaks are connected to edge nodes with a weight of θ. The Δ_k values are connected to the NOT USED node with a weight of α, which can be considered a prior on including the Δ_k in the alphabet. Peaks are connected to each other with a weight of β and connected to the USED node with a weight equal to their intensity. This model discourages removing Δ_k nodes by requiring the cut of multiple θ, β, and p_i values. Connectivity is encouraged through the use of the β weight; to disconnect peaks requires adding β to the cut.

Then each pair of edges which are adjacent to each other (*i.e.* $e^{(0)}_{0,1}$ and $e^{(0)}_{1,3}$) are connected with an edge weight of β. For clarity, the β labels were left off of the figure but all solid green edges have a weight of β.

2.2.2 Edge centric model

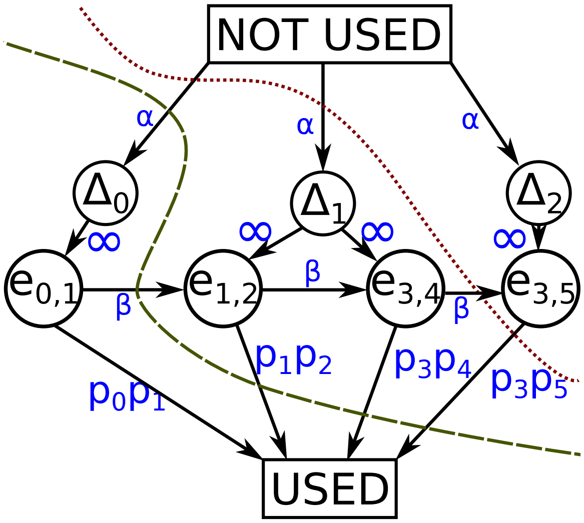


Figure 2.3 **Edge-centric graph-cut model** used to find a minimum alphabet. The dotted dark red line represents a graph cut which would create the alphabet $\Delta = \{\Delta_0, \Delta_1\}$ and the dashed dark green line represents a graph cut which would create the alphabet $\Delta = \{\Delta_0\}$. Large connected graphs are encouraged by the use of the β weighted edges.

A more edge centric model (2.3) does not include any nodes for the peaks. Instead the edge nodes are connected to the USED node with a weight of the product of the peak intensities connected by the edge. This greatly reduces the number of edges and nodes in the graph compared to 2.2.

This model discourages removing Δ_k nodes by forcing a cut through β values and the peak intensity products connecting the edges to the USED node. Larger connected graphs are encouraged through the use of the β edges which have to be cut if two Δ_k values create adjacent edges but only one is to be in the alphabet.

2.2.3 Edge centric model with extra connectedness incentive

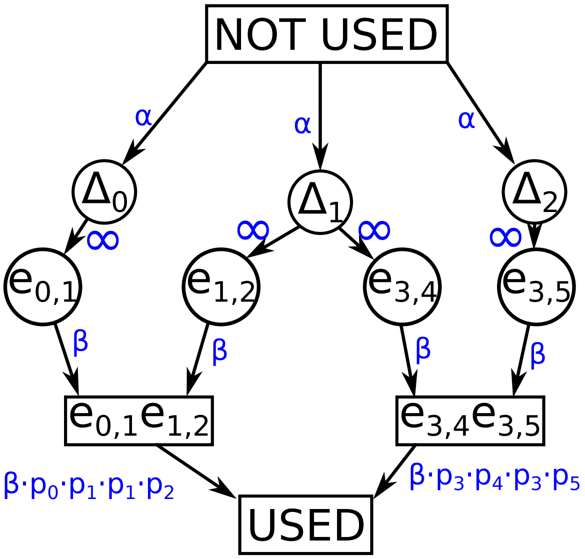


Figure 2.4 **Edge-centric graph-cut with extra connectedness incentive model** used to find a minimum alphabet. Model is similar to 2.3 with an extra layer of nodes between the edge nodes and the USED node. This extra layer is to discourage cutting between two Δ_k nodes which form adjacent edges which connect high intensity peaks.

Another edge-centric model (2.7) was implemented to try and discourage breaking apart Δ_k nodes which form adjacent edges of three high intensity peaks.

2.2.4 Emphasizing Δ-to-Δ edge model

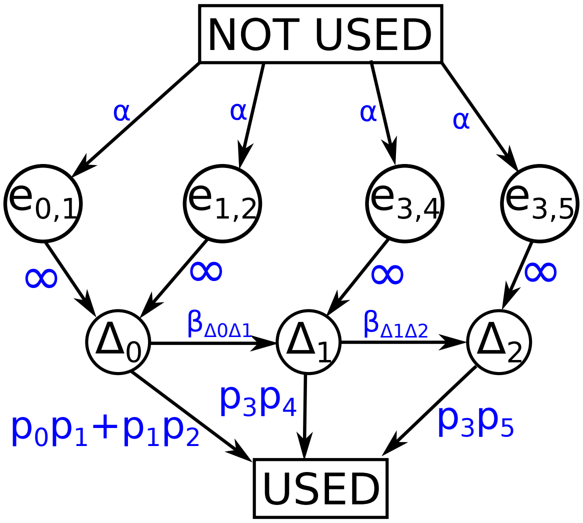


Figure 2.5 **A dynamic Δ-to-Δ edge graph-cut model** used to find a minimum alphabet. In this model the edges connecting Δ node to Δ node are not uniform but are based on how often edges generated by the two Δ values touch each other.

A model with dynamic edges connecting Δ_k nodes is used to try and focus the graph-cut around the Δ_k nodes since they are what creates the alphabet. The edges now have a uniform prior, α, and the Δ_k nodes are connected to USED by the sum of the products of each peak-pair it connects. The edge between the Δ_i and Δ_j is weighted according to the amount of adjacent edges the two form. For each pair of adjacent edges generated by the two nodes, the weight is increased by adding the product of the intensities of the three peaks touched by the edges. This is meant to encourage large connected graphs by making Δ_i, Δ_j pairs harder to cut if they form many adjacent edge pairs. Using one of those Δ_i in the alphabet means the other Δ_j is strongly incentivized to be included.

2.2.5 Energy minimization model

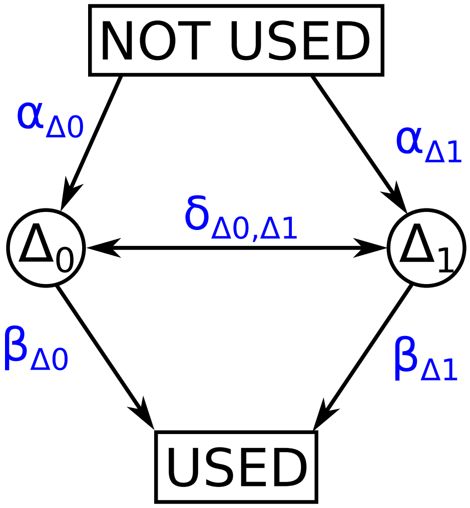


Figure 2.6 **Energy minimization inspired graph-cut model** used to find a minimum alphabet.

This model was inspired by an energy minimization simulation. The weight connecting the Δ_k node to NOT USED is considered the energy included in the system if the Δ_k is used in the alphabet (because cutting it would mean using Δ_k). Similarly, the weight connecting Δ_k to USED would be considered the energy required to not use Δ_k in the alphabet.

The energy to use a Δ_k node is $\sum\limits_{i,j,\ell} p_i p_j$, normalized by dividing by the largest such energy for any Δ_k. The energy to not use Δ_k is calculated so the energy to use and not use Δ_k is equal to one.

The edge between the Δ_i and Δ_j is weighted according to the amount of adjacent edges the two form. For each pair of adjacent edges generated by the two nodes, the weight is increased by adding the product of the intensities of the three peaks touched by the edges. Then, after all Δ-Δ weights have been calculated they are all divided by the average value. This is in order to normalize the values.

2.2.6 Normalized graph-cut model

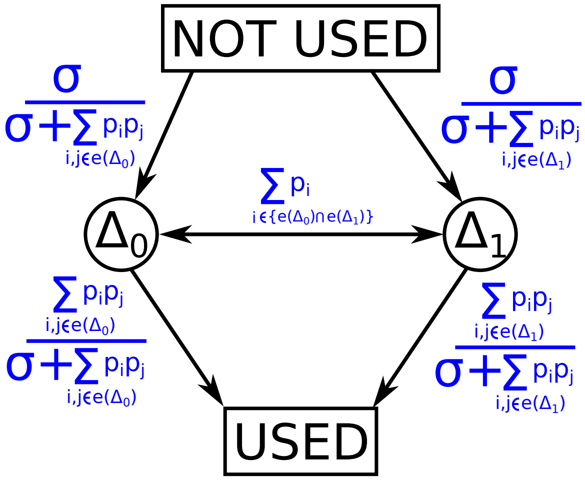

NOT USED
$\frac{\sigma}{\sigma+\sum_{i,j\in e(\Delta_0)} p_i p_j}$
$\frac{\sigma}{\sigma+\sum_{i,j\in e(\Delta_1)} p_i p_j}$
$\sum_{i\in\{e(\Delta_0)\cap e(\Delta_1)\}} p_i$
Δ_0
Δ_1
$\frac{\sum_{i,j\in e(\Delta_0)} p_i p_j}{\sigma+\sum_{i,j\in e(\Delta_0)} p_i p_j}$
$\frac{\sum_{i,j\in e(\Delta_1)} p_i p_j}{\sigma+\sum_{i,j\in e(\Delta_1)} p_i p_j}$
USED

Figure 2.7 **A normalized graph-cut model** used to find a minimum alphabet. The edge weight connecting a Δ_k node to NOT USED is considered the prior. The other edges are also normalized and used to discourage cutting Δ_k from the alphabet.

This model was an attempt to morph the energy minimization model into a normalized probabilistic model. Each Δ_k node is connected to NOT USED with a weight of $\sigma / \left(\sigma + \sum_{i,j\in e(\Delta_k)} p_i p_j\right) \in [0,1]$ and connected to USED with a weight of $\left(\sum_{i,j\in e(\Delta_k)} [p_i p_j\right) / \left(\sigma + \sum_{i,j\in e(\Delta_k)} p_i p_j\right) \in [0,1]$. Similarly, a pair Δ_i, Δ_j are connected (by two edges of opposite direction) with a weight equal to the sum of the intensities of the peaks which are touched by both an edge generated by Δ_i and an edge generated by Δ_j.

2.3 Markov chain Monte Carlo

Each neutral loss alphabet $\Delta = \delta$ is the same constant, given size, d, and deterministically produces a graph consisting of the edges E; these edges connect every pair of peaks within one spectrum if the m/z difference between the peaks is within ϵ of the m/z difference created by dividing alphabet mass Δ_k by charge z:

$$E^{(\ell)}_{z,i,j,k} = \begin{cases} 1 & |m^{(\ell)}_j - m^{(\ell)}_i - \frac{\Delta_k}{z}| \leq \epsilon \\ 0 & \text{else.} \end{cases}$$

The edges E can be found deterministically once Δ and D are known; for this reason,

$$\Pr(D|\Delta = \delta) = \Pr(D|\Delta = \delta, E = e) = \Pr(D|E = e).$$

We assume that all spectra $s^{(1)}, s^{(2)}, \ldots$ (and their masses and intensities) are conditionally independent from one another given the graph induced by E:

$$\begin{aligned} \Pr(D|\Delta = \delta) &= \Pr(D|E = e) \\ &= \Pr(D^{(1)}, D^{(2)}, \ldots |E = e) \\ &= \prod_{\ell} \Pr(D^{(\ell)}|E = e) \\ &= \prod_{\ell} \Pr(s^{(\ell)}, m^{(\ell)}, p^{(\ell)}|E = e). \end{aligned}$$

Conditional independence of the spectra given the edges is fairly reasonable, because it resembles the fact that, given the sample content (which is informed through the graph of connected peaks), the production of one fragmentation spectrum does not interfere with the process by which other fragmentation spectra are produced. Even the caveat, competition between abundant analytes in data-dependent acquisition

(DDA), applies more to which precursors will be selected for fragmentation rather than how peaks in those fragmentation spectra can be connected.

We seek, δ^* a *maximum a posteriori* (MAP) estimate of Δ:

$$\begin{aligned}
\delta^* &= \operatorname*{argmax}_{\delta} \Pr(\Delta = \delta | D) \\
&= \operatorname*{argmax}_{\delta} \prod_{\ell} \Pr(D^{(\ell)} | \Delta = \delta) \cdot \Pr(\Delta = \delta) \\
&= \operatorname*{argmax}_{\delta} \prod_{\ell} \Pr(s^{(\ell)}, m^{(\ell)}, p^{(\ell)} | \Delta = \delta) \cdot \Pr(\Delta = \delta).
\end{aligned}$$

2.3.1 Non-combinatorial approach

The following is a non-combinatorial approach, meaning $\Delta_0, \Delta_1, ...$ do not influence each other and so can be solved for separately rather than as a joint d-dimensional vector Δ. This non-combinatorial approach can be executed by empirically estimating the distribution of m/z differences $m_j^{(\ell)} - m_i^{(\ell)}$ over all spectra ℓ. This can be performed in an unweighted manner (all (i,j) pairs contribute equally to the distribution) or in a weighted manner (an (i,j) pair has contribution proportional to $p_j^{(\ell)} \cdot p_i^{(\ell)}$). Because exactly overlapping differences are improbable, the non-combinatorial approach treats two differences as equal if they are within ϵ of one another. The process of finding all differences $m_j^{(\ell)} - m_i^{(\ell)}$ can be done efficiently using the fast Fourier transform (FFT) by binning the spectrum by m/z then convolving the spectra with itself.

The alphabet $\Delta_1, \Delta_2, \ldots \Delta_d$ is estimated as the top d peaks in the empirical distribution after being sorted by either count in the unweighted case or the sum of the proportional $p_j^{(\ell)} \cdot p_i^{(\ell)}$ values in the weighted case. It is important to note that this non-combinatorial approach only cares about the number of occurrences of the Δ values, does not take into account the connectivity of any graphs which are formed by the edges induced by Δ.

2.3.2 Combinatorial approach

The non-combinatorial approach does not incentivize building of large connected graphs, such as long amino acid chains in a peptide, or large forking substructures in glycoconjugate spectra [7]. A combinatorial approach can be used to incentivize large connected graphs.

2.3.2.1 Efficient graph construction

For each spectrum $D^{(\ell)}$, we efficiently build the graph of all possible connected peaks. In each spectrum $D^{(\ell)}$ and for each charge state z, we create an edge $E^{(\ell)}_{z,i,j,k}$ if and only if

$$|m_j^{(\ell)} - m_i^{(\ell)} - \frac{\Delta_k}{z}| < \epsilon.$$

This connects two peaks whose m/z difference is within ϵ of the predicted m/z difference from alphabet mass Δ_k using charge z.

Of course, for any charge state z and some fixed spectrum ℓ consisting of n peaks, edges can be trivially formed in $\Theta(n \cdot n \cdot d)$; however, by sorting the $m^{(\ell)}$ values and the Δ values, this can be sped up: By proposing the peaks $m_i^{(\ell)}$ and $m_j^{(\ell)}$ first, we know that we're looking for an alphabet mass with $\frac{\Delta_k}{z}$ within ϵ of $m_j^{(\ell)} - m_i^{(\ell)}$; because the search for Δ_k can be processed on the sorted array, this can be accomplished in $\Theta(n \cdot n \cdot \log(d))$ steps. Likewise, if we first propose starting peak $m_i^{(\ell)}$ and alphabet mass Δ_k, then we are searching for the ending peak $m_j^{(\ell)}$ with m/z value within ϵ of $m_i^{(\ell)} + \frac{\Delta_k}{z}$; this can be accomplished in $\Theta(n \cdot d \cdot \log(n))$ steps. This problem is closely related to the famous 3SUM problem (here we have a generalization because it allows matches within ϵ instead of requiring exact matches as the classic 3SUM problem does). Interestingly, there exists no known solution to the classic 3SUM problem in $O(n^{2-\Omega(1)})$ [24]. Furthermore, the "within ϵ" criteria does not easily accomodate use of hashing (used to achieve one $O(n^2)$ algorithm) or other advanced approaches.

In practice, we accelerate the $\log_2$ search for each spectrum by computing a dense table of the cumulative counts of peaks with m/z at or below some target m/z value x. This table has bin widths of α:

$$c_t^{(\ell)} = |\{i \ : \ m_i^{(\ell)} \leq t \cdot \alpha\}|.$$

If $\alpha \geq \epsilon$, we can then use this table to find bounds on indices with which we seed the $\log_2$ search: The lower bound index for matches will be found by $c_{x \cdot \alpha - \alpha}^{(\ell)}$. The upper bound index for matches will be found by $c_{x \cdot \alpha + \alpha}^{(\ell)}$.

Using these bound values, we finish with two $\log_2$ searches: one searches for the first peak with m/z crossing $x - \epsilon$, and the other searches for the last peak with m/z not crossing $x + \epsilon$. In practice, we observe a substantial speedup, even when the number of peaks in the spectrum is relatively few (Table 2.2). This $c^{(\ell)}$ table has the effect of uniformizing the m/z search space; for some distributions of m/z values, this can make the lookup run in constant time.

Furthermore, because the n peaks are stored in contiguous, sorted order (in an array, not a balanced binary search tree), we can define all ending peaks j that would be within ϵ of starting peak i using alphabet mass Δ_k and record them with only two integers: the beginning of the matching window and the size of the matching window. This likewise introduces a considerable speed advantage over using a linked list of peak indices (which would not be cache localized). By choosing a large enough α, constructing a $c^{(\ell)}$ table for fragmentation spectrum ℓ takes space roughly equivalent to the sorted m/z array, $m^{(\ell)}$, and the intensity array, $p^{(\ell)}$.An α which is sufficiently small will create a table which is too large to fit into cache, causing cache misses and slowing the search (this happens for $\alpha = 0.0001$ in table 2.2). Too large of an α can create a large space for the two log searches, similarly slowing the search.

	Alpha	Naïve search	Log search	Binned-log search
Average runtime(s)	0.0001	0.45861	0.08461	0.01541
	0.005	0.45841	0.08472	0.00862
	0.01	0.45902	0.08344	0.00780
	0.02	0.45873	0.08342	0.00712
	0.05	0.45826	0.08483	0.00738
	0.1	0.45838	0.08464	0.00778
	0.5	0.45919	0.08490	0.01304
	1	0.45847	0.08479	0.01820

Table 2.2 **Runtimes to find a peak in a spectrum within $\epsilon = 0.01Da$ of the target m/z value, repeated for 2^{20} such searches on a spectrum with 1,000 peaks.** Note that for $\alpha < \epsilon$ the size of the window returned by the search must be widened to find the correct peak.

As a result of this, on a spectrum of the size in Table 2.2, we get an 11.7-fold speed-up over a standard log search.

2.3.2.2 MAP estimation using sampling

In order to find the best alphabet we could simply try all alphabets. However, for the larger of the two datasets we analyzed there are 58,051,970 possible peak pairs. Then for an alphabet of size d there would be $\binom{58{,}051{,}970}{d} \approx O(58,051,970^d)$ possible alphabets. For an alphabet of size 16 the number of possible alphabets is larger than the number of particles in the universe and this dataset of size 1,891 is relatively small for mass spectrometry where it is not uncommon to have millions of spectra.

Since it is not feasible to try all alphabets, we use a Markov chain Monte Carlo (MCMC) method to approximate the distribution $\Pr(D, \Delta = \delta')$. Specifically, the MCMC method we use is Gibbs sampling[25] because we want to propose one new $\Delta_k | \Delta_1, \Delta_2, \ldots \Delta_{k-1}, \Delta_{k+1}, \ldots \Delta_d$ per iteration. For each univariate cross-section, the changes to Δ_k are proposed and accepted via Metropolis-Hastings [26].

Each Δ_k is proposed from one of three proposal functions (with the choice of proposal function selected at uniform):

1. Select an m/z from the intensity-weighted distribution used for the non-combinatorial approach (selected from all possible m/z differences, not just the top d).

2. Scale Δ_k to have an equivalent m/z value at some charge state. *E.g.*, if $\Delta_k = 3$, it may propose 1 (from $z = 3$ to $z = 1$), 2 (from $z = 3$ to $z = 2$), $\ldots$ or 9 (from $z = 1$ to $z = 3$).

3. Select a random peak in some connected component for some charge state and then chooses a new value for Δ_k that would create a new edge incident to that peak, thereby adding a new edge to the connected component.

The first and third proposal functions are topologically equivalent in that they have the same solution space to pull Δ_k from; however, the third solution is greedy and guarantees that the value it selects will connect a peak to some already existing connected component, the first proposal function does not make this guarantee.

The updated joint probability $\Pr(D, \Delta = \delta')$ is compared with the current joint probability $\Pr(D, \Delta = \delta)$. If $\Pr(D, \Delta = \delta') > \Pr(D, \Delta = \delta)$, then the new $\Delta_k = \delta_k$ is accepted; otherwise the probability of accepting the new $\Delta_k = \delta_k$ is

$$\frac{\Pr(D, \Delta = \delta')}{\Pr(D, \Delta = \delta)}.$$

A value proportional to the joint probabilities can be computed as the product between a prior on Δ and a likelihood proportional to $\Pr(D|\Delta)$.

2.3.2.3 Likelihood model

Here we model the process by which E creates the peaks in spectrum ℓ. We partition $E^{(\ell)}$ into $E_1^{(\ell)}, E_2^{(\ell)}, \ldots$, connected components for each charge state z:

$$\Pr(D^{(\ell)}|E^{(\ell)}) = \prod_z \Pr(D^{(\ell)}|E_z^{(\ell)}).$$

We compute $\Pr(D^{(\ell)}|E_z^{(\ell)})$ as the likelihood of the graph using a particular charge state z. Let $g(E_z^{(\ell)})$ be a collection of the edges in each connected component of the graph formed by $E_z^{(\ell)}$. We define the likelihood of the graph formed when using that particular charge state to be the sum of the likelihoods over these connected components:

$$\Pr(D^{(\ell)}|E_z^{(\ell)}) = \sum_{g \in g(E_z^{(\ell)})} \Pr(D^{(\ell)}|G = g).$$

Lastly, we define the likelihood of a single connected component g using a single charge state z on a single spectrum ℓ using the intensities of the peaks joined by each edge:

$$\Pr(D^{(\ell)}|G = g) = \prod_{(i,j) \in g} p_i \cdot p_j.$$

The values p_i and p_j have been normalized by dividing by the minimal intensity value.

With a simple example one can see how this model motivates an alphabet that seeks to form larger connected graphs instead of many smaller graphs which may have more edges. Lets look at two scenarios, both with only $z = 1$ charge state. The first is a graph with four peaks and four edges. The second is a set of two graphs each with three peaks and two edges. Let all peaks in both examples have the same intensity, p. Then the likelihood of the first scenario is $(p \cdot p) \cdot (p \cdot p) \cdot (p \cdot p) \cdot (p \cdot p) = p^8$.

The second scenario will have likelihood $(p \cdot p) \cdot (p \cdot p) + (p \cdot p) \cdot (p \cdot p) = 2p^4$. For large p, $p^8 \gg 2p^4$.

2.3.2.4 Prior model

The prior model has three requirements, which together produce a prior of either 0 or 1: The first requirement is that all alphabet masses be $\geq 1 - \epsilon$. This restricts alphabets to larger masses; being that smaller masses often have no chemical significance, if we do not enforce this, small masses may be selected because they are actually differences between actual alphabet masses. The second requirement is that no two masses in the alphabet produce similar $m/zvalues$ at any charge considered (for example, $\Delta_1 = 1.00860, \Delta_2 = 2.01720$ would not be possible in the same alphabet). This prevents doubling (or tripling, *etc.*) up on a single alphabet mass strongly supported by the spectra. The third requirement is that no alphabet results be within 0.5Da of one another (*e.g.*, $\Delta_1 = 1, \Delta_2 = 1.1$ would not be possible in the same alphabet).

$$\Pr(\Delta) = \begin{cases} 1 & \forall k, \Delta_k \geq 1 - \epsilon \\ 0 & \text{else} \end{cases} \cdot \prod_{k_1 \neq k_2} \begin{cases} 1 & \forall z_1, z_2, \frac{\Delta_{k_1} \cdot z_1}{\Delta_{k_2} \cdot z_2} \notin [1 - \epsilon, 1 + \epsilon] \\ 0 & \text{else} \end{cases}$$
$$\cdot \prod_{k_1 \neq k_2} \begin{cases} 0 & |\Delta_{k_1} - \Delta_{k_2}| < \frac{1}{2} \\ 1 & \text{else} \end{cases}$$

For faster runtime, we encode the prior model using the random proposal distribution. Given the current alphabet Δ, we propose an alphabet Δ' that is identical in all but one character Δ_k, which has been changed. We do this by first randomly choosing k, the index that will be changed, and then proposing δ'_k a new value for Δ_k.

The new value is proposed by one of the three proposal functions described above.

When exactly one value in the current alphabet (Δ_t) can produce an m/z value too similar to the newly proposed mass (for some charge states z_1, z_2), we could simply reject the proposal as having a zero prior probability; however, that approach can lead to fixation in local optima of the likelihood surface, because it can be difficult to exchange an alphabet mass with a multiple of itself that would produce an equivalent m/z at a different charge state. Instead, it is more efficient to simply assign at index $k = t$ to overwrite Δ_t if the proposal is accepted. When two or more values in the current alphabet can produce an m/z too similar to the newly proposed mass (for some charge states z_1, z_2), then the modification to the alphabet would lower the prior probability to 0; therefore, the proposal is simply repeated without building the graphs or computing the likelihood.

The prior probability is completely accounted for in the proposal step, and thus we may substitute $\Pr(D|\Delta)$ for $\Pr(D, \Delta)$.

2.3.2.5 Adjusting likelihood steepness using θ

In traditional Metropolis-Hastings, a proposal from Δ to Δ' will be accepted with probability

$$\frac{\Pr(D, \Delta')}{\Pr(D, \Delta)},$$

accepting the proposal certainly when $\Pr(D, \Delta') \geq \Pr(D, \Delta)$. We allow for this to be distorted using hyperparameter θ, accepting the proposed change from Δ to Δ' with probability

$$\left(\frac{\Pr(D, \Delta')}{\Pr(D, \Delta)}\right)^{\theta}.$$

The motivation behind including θ is that the MCMC will not mix well if the surface is too steep, and will not find the optimum efficiently if the surface is not steep enough.

In this manner, $\theta = 0$ results in always accepting proposed changes and $\theta = \infty$ results in only accepting changes that immediately improve the joint probability. In the experiments outlined here, we use $\theta = 1$, but offer the ability to set θ to different values at the command line.

Additionally, our software implementation outputs the acceptance rate of proposals as well as the average deviation between $\log(\Pr(D, \Delta'))$ and $\log(\Pr(D, \Delta))$ to help adjust θ. For example, if you want to set θ to get roughly a 50% acceptance rate and you know that the previous run gave an average deviation between $\log(\Pr(D, \Delta'))$ and $\log(\Pr(D, \Delta))$ of x, then $e^x = \left(\frac{\Pr(D,\Delta')}{\Pr(D,\Delta)}\right)$ and you can solve $(e^x)^\theta = 0.5$ for θ.

The same objective could be accomplished using simulated annealing where a loose θ value turns hard according to some carefully selected cooling curve which allows for the most probably outcome to be expected with probability of one if the simulation is ran long enough [27].

Ranking masses in Δ

If desired by the user, using a flag at runtime, the frequency in which masses are in the alphabet may be written to in a file. This may be used to create a ranking of the Δ values based on how many iterations of the Gibbs sampler they stayed in the alphabet. This is done for all masses, not just the masses in the final alphabet.

2.3.2.6 Mapping Δ m/z values to canonical masses

Inferring masses from mass-to-charge gaps is difficult, because two masses may look identical at different charge states. For this reason, the combinatorial approach sometimes finds integer multiples or fractions of a mass instead of the mass itself. For example, water has a mass of roughly 18.01057Da; however, the combinatorial approach may find some $\Delta_k = 36.02114 = 2 \cdot 18.01057$. Generally, if multiple charge states of neutral water losses are well represented, we would expect using $\Delta_k = 18.01057$ will produce a superior likelihood compared to using $\Delta_k = 36.02114$, and therefore the combinatorial approach would eventually choose the canonical mass; however, there are cases where using $\Delta_k = 36.02114$ may produce a higher likelihood. For example, if three peaks indicate a double neutral loss of water peaks a, b, c at xTh, $(x + 18.01057)$Th, and $(x + 36.02114)$Th), then $\Delta_k = 36.02114$ can connect $a \rightarrow c$ using a charge state of $z = 1$ and also connect $a \rightarrow b$ and $b \rightarrow c$ using a charge state of $z = 2$. If the $z = 3$ charge state is not well represented (using $\Delta_k = 36.02114$ will not find gaps of size 9.0075Th produced by water at a charge state of $z = 3$), then the model will prefer $\Delta_k = 36.02114$ to $\Delta_k = 18.01057$.

For this reason, before we report the final mass alphabet Δ, for each $\Delta_k \in \Delta$, we compare the masses $\frac{\Delta_k}{1}, \frac{\Delta_k}{2}, \frac{\Delta_k}{3}, ..., \frac{\Delta_k}{c}$ where c is the value of the max charge used in the Gibbs sampler. For each of the new candidate masses, Δ'_k, the graphs produced over all spectra for its charge states $\frac{\Delta'_k}{z=1}, \frac{\Delta'_k}{z=2}, \frac{\Delta'_k}{z=3}, ..., \frac{\Delta'_k}{z=c}$ are built. If $\frac{\Delta_k}{1}$ and its charge states produce the most edges, we report the mass as Δ_k (unchanged); if $\frac{\Delta_k}{2}$ and its charge states produce the most edges, we report the mass as $\frac{\Delta_k}{2}$. In this manner, double neutral losses, double mass differences, and dimers do not force us to report multiples or fractions of the mass of interest.

2.3.3 Finding recurring structures via similar subgraphs

Given the Δ collection estimated by the Gibbs sampler, we are able to use the *de novo* approach to connect as many peaks as possible in each spectrum at every charge state of interest. On each spectrum and for each charge state, we record all connected components.

From this collection of graphs, we would like to find large connected components that are isomorphic to one another (*i.e.*, one graph is the same as the other, but with renamed vertices); however, graph isomorphism is a difficult problem: although it is not known if it is NP-complete, it is thought to be recalcitrant enough to be employed in cryptography [28].

For this reason, finding large, recurring structures in the *de novo* graphs appears difficult. This is made more difficult if we generalize to the optimization variant in which we find the largest isomorphic subgraphs of each graph, rather than scoring each as “isomorphic” or “not isomorphic”.

2.3.3.1 Finding graph isomorphism with cross-correlation of subspectra

Fortunately, the graphs that we are using have a metric property in which distances are preserved. For instance, if a graph connects peaks at 2Th, 6Th, and 9Th, then any isomorphic graph must connect peaks of the form xTh, $(x+4)$Th, and $(x+7)$Th (*e.g.*, 90Th, 94Th, and 97Th). For this reason, we can use cross-correlation of the subspectra (*i.e.*, the peaks that correspond to nodes in our graph) to discover the largest isomorphic subgraph. The cross-correlation shifts the two subspectra over one another and computes the dot product at each shift. The shift that produces the maximum dot product solves for x, and the peaks that align at that shift indicate corresponding nodes in the two subgraphs.

Using this approach, we can efficiently score pairs of connected components for similarity.

2.3.3.2 A locality-sensitive hashing approach to clustering subgraphs

We could use this cross-correlation approach to find the largest isomorphic subgraphs on all pairs of connected components found in all spectra; however, the runtime of this would be quadratic in the total number of connected components found (and this would be far more than quadratic in the number of spectra); this is not efficient enough to be applied to many spectra.

For this reason, we generalize locality-sensitive hashing (LSH) to find subspectra that have a high maximum value in the cross-correlation (the maximum value of the cross-correlation is the measure of subgraph isomorphism described immediately above).

LSH encodes objects (*i.e.*, subspectra) as large vectors by binning them by m/z. The probability that a random plane cuts between two such vectors is $1 - \frac{\psi}{\pi}$, where ψ is the angle between the two vectors [29, 30]; therefore, by applying a random plane to an object, we get 1 bit of information for that object (*e.g.*, a 0 is encoded by being on the negative side of the vector normal to the plane, a 1 is encoded by being on the positive side of the vector normal to the plane). We can apply this procedure b times, thereby producing a b-length bit-string label for each object, and thus binning each object into one of 2^b bins. If several planes are applied, there is only a small probability that two dissimilar objects would reach the same bin. This has recently been applied to clustering mass spectra [31].

This standard LSH approach to clustering mass spectra cannot be applied in our case because we do not know the shift between a pair of subspectra that would allow them to align and produce a high dot product; LSH does not work in this case.

We introduce a means by which we can cluster spectra that allows spectra to be placed into a similar bin even when they are shifted. Given a vector a (from binning a spectrum) and a vector b (from binning a second spectrum) where both have length

n, we note the value of index k for each discrete Fourier transform (DFT):

$$\begin{aligned} A_k &= \sum_{i=0}^{n-1} a_i \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \\ B_k &= \sum_{i=0}^{n-1} b_i \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}. \end{aligned}$$

We note $a \equiv b$, *i.e.*, a is equivalent to b up to rotation, if $\exists u : a_{(i+u) \bmod n} = b_i$. Thus we have

$$\begin{aligned} B_k &= \sum_{i=0}^{n-1} a_{(i+u) \bmod n} \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \\ &= \sum_{i=0}^{n-1} a_i \cdot e^{-(i-u) \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}, \end{aligned}$$

because we can equivalently shift the a_i terms forward or the $e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}$ terms backward by u. Thus

$$\begin{aligned} B_k &= \sum_{i=0}^{n-1} a_{(i+u) \bmod n} \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \cdot e^{u \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \\ &= A_k \cdot e^{u \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}, \end{aligned}$$

i.e., rotating a sequence will simply change the phases of each index of the DFT.

If we ignore the phase of each term in the DFT (using the magnitudes $|A_k|$ and $|B_k|$ at each index, known in signal processing as the "power spectra"), then two objects that are identical up to rotation must look identical.

Thus, we use fast Fourier transform (FFT) [32] to create the power spectrum of each subspectrum derived from a connected component, and then use LSH to bin similar power spectra. Bins that contain subspectra coming from many large connected graphs are indicative of *de novo* results that are likely reproduced in multiple spectra and multiple charge states. These recurring subgraphs give insight into common chemical structures found with the inferred alphabet Δ (Figure 2.8).

Importantly, the cost of running the above procedure (ignoring the cost of performing the FFT for each subspectrum corresponding to a connected component) will be linear in the number of connected components investigated, an improvement from quadratically many computationally difficult graph isomorphism problems.

2.4 Edge-maximal isomorphic subgraphs on embeddings in $\mathbb{Z}^d$

Let $G = (V, E)$ be a Euclidean graph (or, alternatively, a weighted Euclidean graph) embedded into $\mathbb{Z}^d$. Excluding the degenerate case where vertices have 0 distance to one another, each vertex must correspond to a unique location in $\mathbb{Z}^d$ such that the distance between any pair of vertices in the graph (using number of edges if G is unweighted and weighted distance if G is weighted) is preserved by the distances of the vertex locations in the embedding. Let this embedding take the form of a function f from nodes to $\mathbb{Z}^d$.

Give an f to perform the embedding, the edges of the graph can be encoded as a tensor $T(G) \in \mathbb{Z}^d \times \mathbb{Z}^d = \mathbb{Z}^{2d}$ as an adjacency tensor:

$$T(G)_{f(a),f(b)} = \begin{cases} 1, & (a,b) \in E \\ 0, & \text{else.} \end{cases}$$

We say that two embeddings, $T(G_1)$ and $T(G_2)$, are "oriented" with respect to one another if the axes and directions of the embeddings are aligned. Thus, any isomorphic subgraphs $g_1 = (v_1, e_1), g_2 = (v_2, e_2)$ of G_1, G_2 (respectively) will have vertices shifted from one another. Thus any oriented isomorphic subgraph will have $\exists s = \in \mathbb{Z}^d$ such that all matched nodes in the isomorphism align in space: $\forall x \in$

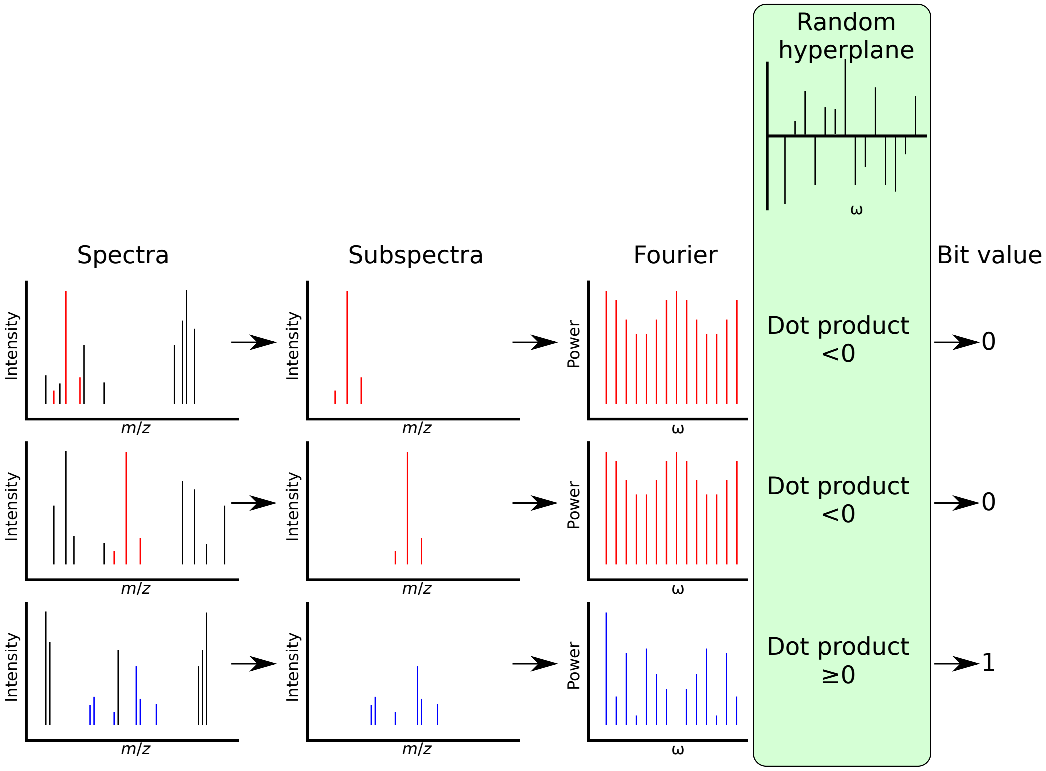


Figure 2.8 **An LSH approach to finding similar subgraphs.** In the left column, three spectra are shown with the subspectra (shown in color), which are peaks contained in a connected component produced by building the graph with the estimated mass alphabet Δ. The second column shows only those peaks in the subspectrum. The third column shows the absolute values of the DFTs of the subspectra. Each of these power spectra is dot producted with a random hyperplane, and the sign of the resulting value is used to produce a single bit. When two connected components have large subgraphs isomorphic to one another, their subspectra must be shifted versions of each other, and thus their power spectra must be nearly identical. Two subspectra drawn (first and second rows) are similar in this manner, producing similar power spectra and thus a low probability of being separated by a random hyperplane. Repeating this process with several different random hyperplanes and concatenating the bits produces a hash, which has a high probability of binning together connected components that have substantial subgraph isomorphism.

$v_1, \forall y \in v_2, \in \mathbb{Z}^d : y = x + (s, s)$. Note that the edge embeddings are $\in \mathbb{Z}^{2d}$ and so we shift by (s, s) because both start and end vertices must be shifted identically to preserve the graphs.

Given these oriented embeddings, the edge-maximal isomorphic subgraph can be found via the cross-correlation between $T(G_1)$ and $T(G_2)$. Let $C = T(G_1) \circledast T(G_2)$ denote the cross-correlation between $T(G_1$ and $T(G_2)$. For any shift $s \in \mathbb{Z}^d$,

$$\begin{aligned} C_{(s,s)} &= \sum_{w,y:w+(s,s)=y} T(G_1)_w \cdot T(G_2)_y \\ &= \sum_w T(G_1)_w \cdot T(G_2)_{w+(s,s)}. \end{aligned}$$

Thus $C_{(s,s)}$ is a dot product of the shifted adjacency tensor $T(G_1)$ on the adjacency tensor $T(G_2)$. Because the tensors have 1 where edges exist and 0 where edges do not exist, each shifted dot product counts the number of edges bijective to one another when $T(G_1)$ is shifted by s relative to $T(G_2)$.

By the definition of orientedness, each considered isomorphic subgraph must have some s whereby $T(G_1)$ has been shifted by s to align the vertex locations with their corresponding vertices in $T(G_2)$. Thus the edge-maximal isomorphic subgraphs must be found at some shift s^*, at which C_{s^*,s^*} will be maximal. Conversely, the entry s^* at which C_s is maximized corresponds to the shift with which the isomorphic subgraph with the largest number of edges is found.

Thus the shift producing the node bijection with greatest number of matching edges can be found by computing C by performing FFT convolution in Z^{2d} and searching C for the index of form (s^*, s^*), at which it attains maximal value. Likewise, given this shift, the matching nodes can be found by locating the nonzero entries in the dot product between the tensors.

2.4.1 Exploiting sparsity

Storing edge embeddings in $\mathbb{Z}^{2d}$ can requires substantial space and performing convolution in $\mathbb{Z}^{2d}$ can require substantial time. We can simultaneously exploit a sparsity in both the edge embeddings and in the cross-correlation, in which we only visit indices of the form (s, s) where $s \in \mathbb{Z}^d$.

We perform this by first transforming the tensor embeddings from starting vertex and ending vertex pairs into bijective (and orientation-preserving) embeddings on vertex differential and starting vertex pairs. That is,

$$T'(G)_{f(b)-f(a),f(a)} = \begin{cases} 1, & (a,b) \in E \\ 0, & \text{else.} \end{cases}$$

$$\begin{aligned} C'_s &= \sum_{(w,x),(y,z):(w+s,x+s)=(y,z)} T(G_1)_{(w,x)} \cdot T(G_2)_{(y,z)} \\ &= \sum_{(w,x)} T(G_1)_{(w,x)} \cdot T(G_2)_{(w+s,x+s)} \\ &= \sum_{(w,x)} T'(G_1)_{(x-w,w)} \cdot T'(G_2)_{(x-w,w+s)} \\ &= \sum_{a} T'(G_1)_a \circledast T'(G_2)_a. \end{aligned}$$

That is, we count the bijective edges at shift $s \in \mathbb{Z}^d$ by counting the edges (via dot product) that have the same differential and the same start location after applying the shift. This can be performed by summing several cross-correlations. These cross-correlations are on tensors $\in \mathbb{Z}^d$; furthermore, the edge embeddings themselves can now be stored in a sparse manner by using a dictionary of edge differentials to vertex start locations. Storage will thus be reduced to $\mathbb{Z}^d$ when the graphs are sparse, but without preventing the benefit of FFT tensor convolution, which can only be applied on arrays.

Thus, when considering a pair of tensor embeddings $T'(G_1)$ and $T'(G_2)$, we can find the isomorphism with the largest number of matching edges by finding the s^* at which C'_{s^*} is maximal.

2.4.2 A shift-invariant LSH encoding

When considering n graphs, we can use the FFT-convolution-based approach above to find pairs of graphs with large edge-maximal isomorphic subgraphs by simply trying all $\binom{n}{2}$ graph pairs; however, when n is large, the number of pairs will become too large to efficiently use this approach.

In order to reduce the number of pairwise matchings we use LSH to first bin the graphs and only compare all pairs in the same bin. LSH relies on each graph being represented as a vector in $\mathbb{R}^m$ such that graphs with a large number of isomorphic edges must be close to one another. LSH exploits the fact that any two points in $\mathbb{R}^m$ will share a plane with the origin, and thus the normalized points can be thought of as separated by an angle ψ on the unit sphere (sampled by sampling from several independent Gaussians and normalizing so that $\| \cdot \|_2 = 1$). The probability that a random vector on the unit sphere will have dot products with different sign is $1 - \frac{\psi}{\pi}$, because the dot product will ignore any axes not in the plane with the origin[29, 30]. Thus, several such random vectors on the sphere can be generated to form a hash for each object: a negative dot product with the random vector will append a 0 bit to the object's hash and a nonnegative dot product will append a 1 bit to the object's hash. Thus, k random vectors will produce k-bit hashes for each object in $O(n \cdot k \cdot m)$ time (which will be faster than n^2 when the number of graphs $n \gg k \cdot m$). Objects with a small angle between them have a much higher probability of hashing into the same bin. Thus, by performing a few replicates of this strategy, the probability that near neighbors do not reach the same bin in at least one replicate becomes small; meanwhile, the expected number of objects that land in the same bin remains small, and thus few pairs must be compared.

The adjacency tensors themselves are not suitable for direct hashing because shifted graphs will have a large angle between them using the Z^{2d} representation; however, the fact that each graph pair can be compared using convolution indicates that they will have significant overlap in the frequency domain.

Given two vectors a and b, we write their frequency domain representations as follows:

$$\begin{aligned} A_k &= \sum_{i=0}^{n-1} a_i \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \\ B_k &= \sum_{i=0}^{n-1} b_i \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}. \end{aligned}$$

We note $a \equiv b$, *i.e.*, a is equivalent to b up to rotation, if $\exists u : a_{(i+u) \bmod n} = b_i$. Thus we have

$$\begin{aligned} B_k &= \sum_{i=0}^{n-1} a_{(i+u) \bmod n} \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \\ &= \sum_{i=0}^{n-1} a_i \cdot e^{-(i-u) \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}, \end{aligned}$$

because we can equivalently shift the a_i terms forward or the $e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}$ terms backward by u. Thus

$$\begin{aligned} B_k &= \sum_{i=0}^{n-1} a_{(i+u) \bmod n} \cdot e^{-i \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \cdot e^{u \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}} \\ &= A_k \cdot e^{u \cdot k \cdot \frac{2\pi}{n} \sqrt{-1}}, \end{aligned}$$

i.e., rotating a sequence will simply change the phases of each index of the DFT.

If we ignore the phase of each term in the DFT (using the magnitudes $|A_k|$ and $|B_k|$ at each index, known in signal processing as the "power spectra"), then two objects that are identical up to rotation must look identical.

Thus, we use fast Fourier transform (FFT)[32] to create the power spectrum of each adjacency tensor derived from a connected component, and then use LSH to bin similar power spectra. Importantly, the cost of running the above procedure (ignoring the cost of performing the FFT for each graph in the set) will likely be $\in o(n^2)$ in the number of graphs investigated, an improvement from quadratically many computationally difficult graph isomorphism problems.

By observing where the FFT would be employed in the cross-correlation between the T' tensor embeddings, we transform the embedding for graph i into the frequency domain via the d-dimensional FFT and then converting to a power spectrum:

$$F(G)_a = |FFT(T'(G_i)_a)|.$$

We perform LSH for this graph by sampling several random Gaussians $g_{a,\ell}\ \mathcal{N}(0,1)$:

$$bit\_hash(G_i) = \begin{cases} 1, & \sum_a \sum_\ell g_{a,\ell} \cdot |FFT(T'(G_i)_a)|_\ell \geq 0 \\ 0, & \text{else.} \end{cases}$$

A k-bit hash for each graph is completed by concatenating k bit_hashes. Note that the Gaussians should be sampled once for each all objects to receive a single bit hash and resampled for each subsequent bit concatenated by the LSH.

CHAPTER 3 RESULTS

3.1 Data

All results were obtained from running the models on the following two datasets.

3.1.1 Manually curated glycoconjugate spectra from human urine

Thousands of glycoconjugate spectra from human urine were manually curated by an expert to find 62 with strong evidence of glycoconjugates [7]. *A priori*, four sugar residue masses (Hexose, HexNAc, dHex, and NeuAc), as well as the neutron mass (whose mass is roughly the shift to produce isotope peaks) are the only masses we expect. Note that these masses were not provided for analysis, but are only used to validate the resulting masses found.

3.1.2 Horseradish peroxidase glycoprotein standard spectra

Glycoprotein stain (Pierce Glycoprotein Staining Kit, catalog number 24562) containing horseradish peroxidase (UNIPROT accession P00433[33]) was analyzed on an ABSciex Triple TOF 5600+, producing 1,891 fragmentation spectra (similarly to [34]).

3.2 Convex optimization

All results from this section were obtained by running the models only on the glycoconjugate data.

3.2.1 Minimal quadratically constrained linear program (M2.1.3)

n	d	Correct	Incorrect	Unique	Time(s)
10	3	3	0	3	10.011
10	5	3	0	3	17.342
13	5	4	0	4	31.548
12	16	0	16	0	95.159
15	16	0	16	0	172.383

Table 3.1 Runtimes and outcome from M2.1.3 when using Mathematica. n is the number of peaks in the spectrum and d is the maximum size of alphabet allowed. Correct is the number of alphabet masses which fit between two peaks within $\epsilon - 0.1$. Unique is the amount of the correct answers which were unique masses.

3.2.2 Lagrangian relaxation (M2.1.4)

n	d	Iterations	Step-size	Correct	Incorrect	Unique	Edges	Time(s)
5	10	10	0.1	5	2	4	9	48.148
10	10	10	0.1	9	0	6	101	324.671
30	16	10	0.1	-	-	-	-	835.197

Table 3.2 Runtimes and outcome from M2.1.4. n is the number of peaks in the spectrum and d is the maximum size of alphabet allowed. Correct is the number of alphabet masses which fit between two peaks within $\epsilon - 0.1$. Unique is the amount of the correct answers which were unique masses.

3.2.3 Minimizing over indicator variables (M2.1.5)

$\|\Delta\|_2$	$\sum\limits_k b_k$
25.12	20.02
18.65	17.95
20.22	16.78

Table 3.3 Alphabets when ran with $\|\Delta\|_2$ in the objective function versus $\sum\limits_k b_k$ in the objective function. There were 10 peaks with an alphabet of size 3.

n	d	Iterations	Step-size	Correct	Incorrect	Unique	Edges	Time(s)
4	5	10	0.1	2	2	2	2	29.954
10	5	10	0.1	0	5	-	-	99.384
10	10	10	0.1	1	9	1	2	291.228
15	15	0	0.1	0	0	0	0	error

Table 3.4 Runtimes and outcome from M2.1.5. n is the number of peaks in the spectrum and d is the maximum size of alphabet allowed. Correct is the number of alphabet masses which fit between two peaks within $\epsilon - 0.1$. Unique is the amount of the correct answers which were unique masses.

3.2.4 An edge-centric model: incentivizing larger graphs (M2.1.6)

n	d	Iterations	Step-size	Correct	Incorrect	Unique	Edges	Time(s)
5	10	10	0.1	1	1	1	3	50.456
10	10	10	0.1	6	0	6	7	321.131
15	10	0	0.1	0	0	0	0	error

Table 3.5 Runtimes and outcome from M2.1.6. n is the number of peaks in the spectrum and d is the maximum size of alphabet allowed. Correct is the number of alphabet masses which fit between two peaks within $\epsilon - 0.1$. Unique is the amount of the correct answers which were unique masses.

3.2.5 Maximize number of edges and finding unique masses through constraints (M2.1.9)

n	d	Correct	Incorrect	Unique	Edges	Time(s)
10	5	5	0	5	5	0.17
15	5	5	0	5	7	0.46
20	5	5	0	5	13	11.23
30	5	5	0	5	25	stopped at 11.16
10	10	10	0	10	10	59.01

Table 3.6 Runtimes and outcome from M2.1.9. n is the number of peaks in the spectrum and d is the maximum size of alphabet allowed. Correct is the number of alphabet masses which fit between two peaks within $\epsilon - 0.1$. Unique is the amount of the correct answers which were unique masses.

3.2.6 Two step method of finding the best Δs and then minimizing the alphabet (M2.1.10)

n	d	Correct	Incorrect	Unique	Edges	Time(s)
10	5	5	0	5	7	0.14
20	5	5	0	5	20	0.329
50	5	5	0	5	65	11.393
100	5	5	0	5	70	180.10

Table 3.7 Runtimes and outcome from M2.1.10. n is the number of peaks in the spectrum and d is the maximum size of alphabet allowed. Correct is the number of alphabet masses which fit between two peaks within $\epsilon - 0.1$. Unique is the amount of the correct answers which were unique masses.

3.2.7 Maximum vertex cover approach (M3.2.7)

Mass value
0.91967
1.90000
2.91732
3.87359
23.8847
35.91240
71.98874
83.86975

Table 3.8 Output alphabet of size 8 using the bipartite graph model (M3.2.7). The program took 27.08 minutes to run on all unfiltered spectra from the glycoconjugate dataset.

3.3 A max-flow/min-cut formulation

For all graph cut models we were unable to get any useful results. They would all almost always choose an alphabet of either zero masses or all masses available. The energy minimization model and the normalized model would sometimes choose an alphabet in between but that would still be all but just a handful of delta values, resulting in an alphabet of several hundred masses.

3.4 Markov chain Monte Carlo

The values in the results are reported using five decimal places, despite having machine tolerances of 0.02Da and 0.05Da. The reason for this is that we often find masses to a much higher precision. This is because if we have a set of masses which are within machine tolerance of the monoisotopic mass of water and connect at least one pair of peaks in a spectrum, then the distribution of the masses in the set should

Rank	Δ	Frequency	label
1	42.01047	16000	
2	84.02204	16000	
3	188.01611	16000	
4	130.00746	15997	
5	0.98410	15953	Neutron/Deamidation
6	18.00746	15952	Water
7	162.04746	15905	Hexose
8	94.03555	15894	

Table 3.9 **Results from ranking masses in Δ for 62 glycoconjugate spectra.** This table shows the rankings of the masses by frequency of presence in Δ. The higher the frequency, the more times this mass (or a mass within ϵ of it) was included in the alphabet. This was run with $\epsilon = 0.02Da$ and $d = 8$.

center around the true monoisotopic mass of water. For example, in the alphabet for the 62 expert-curated spectra which uses $\epsilon = 0.02Da$ we find water at a mass of 18.01068Da which is 0.000115Da from the monoisotopic mass of water and we find a mass of 30.01058Da which is accurate to the value of a serine/glycine substitution, 30.010565Da, for four digits [35].

Ranking masses in Δ

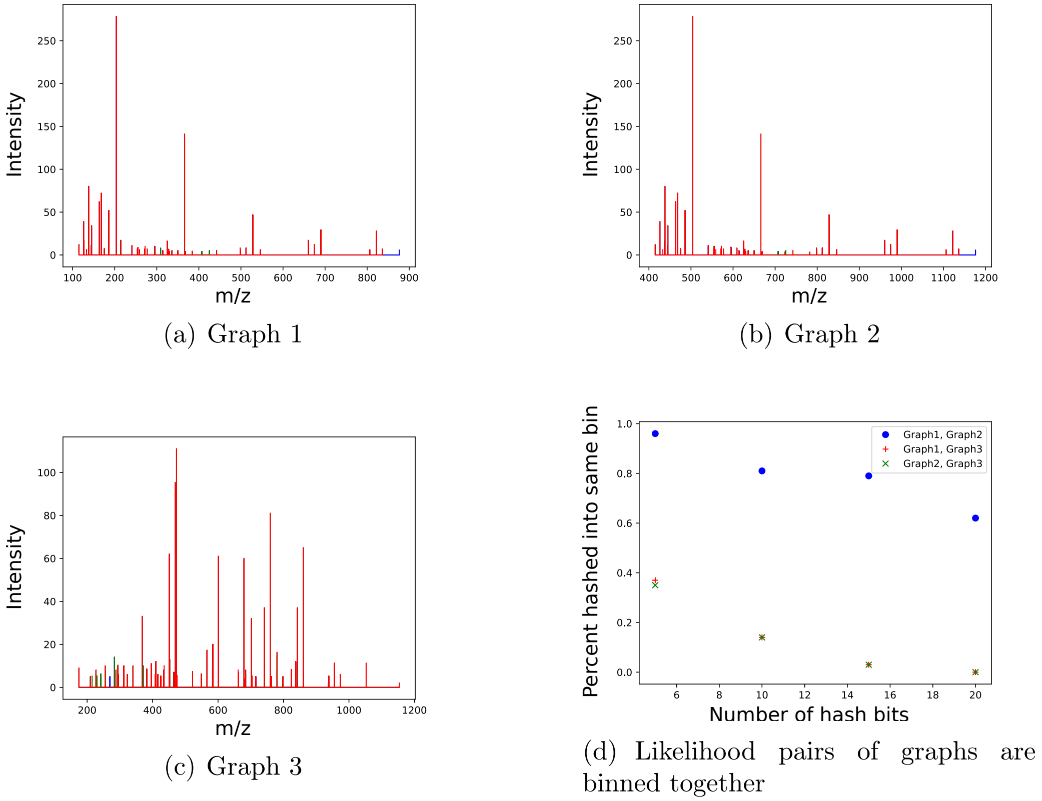
Tables 3.9 and 3.10 show the rankings of masses in alphabets for the 62 glycoconjugate spectra and 1,891 glycoprotein spectra, respectively. In both instances the Gibbs sampler was ran for 16,000 iterations. Taking into account the alphabet sizes for the two tables (8 and 16, respectively) you can see which masses were highly desirable. With some masses were in almost every iteration they must have been proposed early, this shows why having a great proposal function is crucial. These rankins are saved to file before the mapping to canonical mass step.

Rank	Δ	Frequency	label
1	162.05000	16000	Hexose
2	228.07500	15997	2× N
3	0.98210	15996	Neutron/Deamidation
4	18.01130	15986	Water
5	42.00810	15909	
6	30.02500	15899	
7	180.06330	15756	
8	57.00000	15721	G
9	23.00420	15692	
10	144.06510	15650	
11	790.37500	15648	
12	202.10000	15590	
13	17.01790	15569	
14	720.25740	13857	
15	2.07260	9725	
16	839.37500	8935	

Table 3.10 **Results from ranking masses in Δ for 1,891 glycoprotein spectra.** This table shows the rankings of the masses by frequency of presence in Δ. The higher the frequency, the more times this mass (or a mass within ϵ of it) was included in the alphabet. This was run with $\epsilon = 0.05Da$ and $d = 16$.

Efficiency of LSH when hashing pairs of similar and dissimilar graphs

Now, we look at how effective this LSH method is at putting a pair of similar, but shifted, graphs into the same bin versus a pair of very different graphs (figure 3.1). *De novo* sequencing was performed on spectra taken from the 1,891 glycoprotein data set with the alphabet from table 3.14. The graphs are bijective to the subspectra and are created by isolating the peaks connected by the alphabet. Each node in a graph represents a peak in a spectrum, with an edge connecting two nodes in the graph if the peaks in the spectrum are connected by a mass in the alphabet. Graphs 1 and 2 are very similar, but not exactly the same, and are shifted by roughly 300Da. Graph 3 is almost completely different from the first two. For hashes with different number of bits (*i.e.* different number of cutting planes) all pairs were binned together at a different rate with the pair of similar graphs always being binned together at a significantly higher rate than any pair involving the dissimilar graph.



(a) Graph 1

(b) Graph 2

(c) Graph 3

(d) Likelihood pairs of graphs are binned together

Figure 3.1 **Effectiveness of LSH on binning together pairs of similar, but shifted, graphs and pairs of dissimilar graphs.** Three subspectra were created by applying *de novo* sequencing on the 1,891 glycoprotein spectra with the alphabet from table 3.14. Graphs 1 and 2 are very similar subspectra (44 out of 55 similar peaks) but are shifted by roughly 300Da. Graph 3 is a very different subspectra from graphs 1 and 2. In subfigure 3.1(d) the percentage of times each pair of graphs are binned together is plotted versus the number of bits in each hash. In the subspectra, the different colored peaks represent being connected by Δ_k/c values of different charge.

Below we look at the results from two datasets; both used 32 threads. The manually curated glycoconjugate dataset has 62 spectra and was run with $\epsilon = 0.02Da$. The horseradish peroxidase glycoprotein has 1,891 spectra and was run with $\epsilon = 0.05Da$. Both datasets are available at the site listed in the "available" section. The ϵ values are machine-dependent and were recommended by the scientists who produced the data (Dr. Froehlich for the glycoconjugate dataset and Dr. Shu and Dr. Yang for glycoprotein dataset). In each fragmentation spectrum, we remove peaks that are below 1% of the maximum intensity in that spectrum.

3.4.1 Manually curated glycoconjugate spectra from human urine

Thousands of glycoconjugate spectra from human urine were manually curated by an expert to find 62 with strong evidence of glycoconjugates [7]. *A priori*, four sugar residue masses (Hexose, HexNAc, dHex, and NeuAc), as well as the neutron mass (whose mass is roughly the shift to produce isotope peaks) are the only masses we expect. Note that these masses were not provided for analysis, but are only used to validate the resulting masses found. A more detailed explanation of the sample preparation is available in [7].

Non-combinatorial results are shown with $d = 8$ for both the unweighted and weighted approaches (Table 3.11).

The combinatorial approach was run for 16 epochs per thread. Each epoch used 1000 iterations. The total real runtime of the analysis was 4 minutes. Combinatorial approach alphabet results are shown with $d = 8$ (Table 3.12).

Examples of recurring structures found using LSH with the $d = 8$ alphabet projection (*i.e.*, the alphabet reported in Table 3.12) are shown in Figure 3.2.

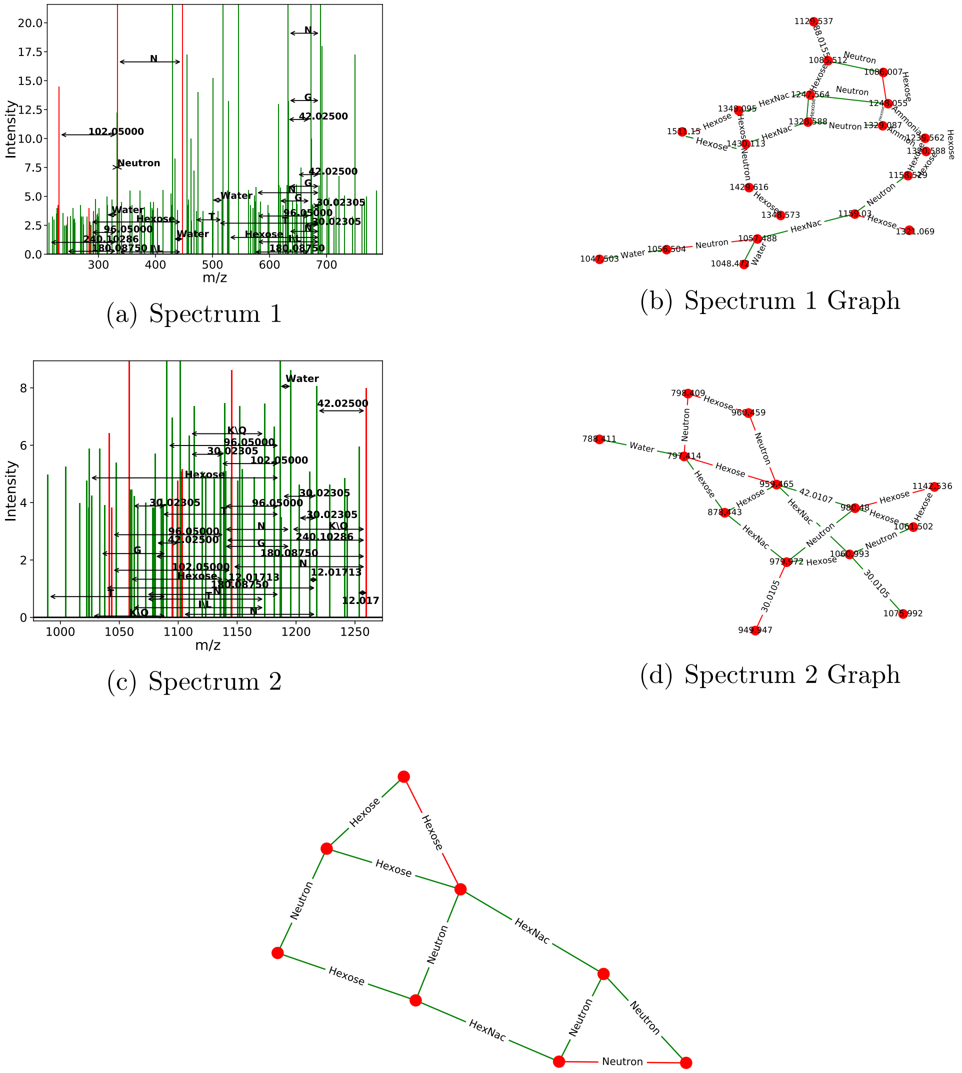
Rank	Mass	Molecule
1	0.99686	Neutron/Deamidation
2	18.00686	Water
3	0.49686	
4	60.01686	
5	42.00686	
6	162.04686	Hexose
7	27.98686	
8	36.01686	
16	17.01686	Ammonia
110	203.07686	HexNAc
923	146.06686	dHex
1,765	291.09686	NeuAc

Rank	Mass	Molecule
1	0.99686	Neutron/Deamidation
2	18.00686	Water
3	0.49686	
4	162.04686	Hexose
5	60.01686	
6	88.00686	
7	36.01686	
8	30.00686	
17	17.01686	Ammonia
136	203.08686	HexNAc
832	146.06686	dHex
1,522	291.09686	NeuAc

Table 3.11 Most frequent $d = 8$ gap pairs ($i.e.$, $m_j - m_i$) on 62 expert-curated glycoconjugate spectra. The left table ranks using the unweighted frequency of gaps and the right table weights each gap by the product of peak intensities $p_i \cdot p_j$. Masses are rounded to 5 decimal points.

Mass value	Manual interpretation	Known *a priori*?	Monoisotopic mass
1.00328	Neutron	Yes	1.00860
17.00746	Ammonia	No	17.02655
18.01068	Water	No	18.01057
30.01058			
42.01071			
88.01555			
162.04746	Hexose	Yes	162.05282
203.06746	HexNAc	Yes	203.07943

Table 3.12 Results when running the combinatorial approach on 62 expert-curated glycoconjugate spectra with $d = 8$. Because the combinatorial approach assigns no ranks to the masses, they are reported in ascending order. Masses are rounded to 5 decimal points. Masses known *a priori* are labeled; these masses were not provided to the model, but instead are known true positives in advance.



(a) Spectrum 1

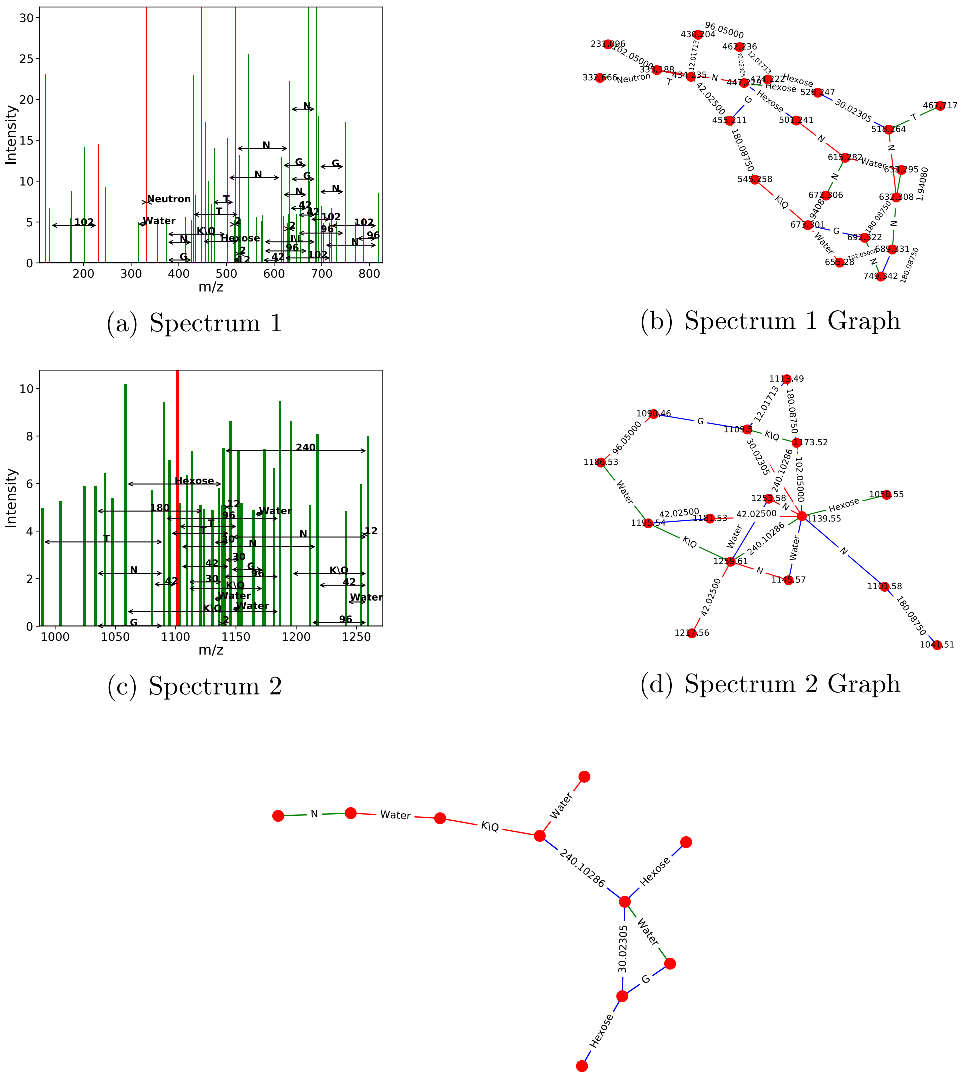
(b) Spectrum 1 Graph

(c) Spectrum 2

(d) Spectrum 2 Graph

(e) Graph of Common Subgraphs

Figure 3.2 **Example similar subgraph pair found using LSH on results from 62 expert-curated glycoconjugate spectra.** Two spectra (a,c) and their corresponding *de novo* graphs (b,d) found using the combinatorial approach. Spectra are drawn with peaks used in the graph colored red and unused peaks colored green. LSH is used to find this matching pair, and fast convolution finds the largest isomorphic subgraph in the pair (e). A minimal amount of peaks were removed from (b,d) for legibility. The top subspectrum is from “120810_JF_HNU142_16.5710.5710.3” and the bottom is from “120810_JF_HNU142_16.6444.6444.4.”



(a) Spectrum 1

(b) Spectrum 1 Graph

(c) Spectrum 2

(d) Spectrum 2 Graph

(e) Graph of Common Subgraphs

Figure 3.3 **Example similar subgraph pair found using LSH on results from 1,891 glycoprotein standard spectra.** Two spectra (a,c) and their corresponding *de novo* graphs (b,d) found using the combinatorial approach. Spectra are drawn with peaks used in the graph colored red and unused peaks colored green. LSH is used to find this matching pair, and fast convolution finds the largest isomorphic subgraph in the pair (e). Some peaks were removed from (b,d) for legibility. The top subspectrum is from “Locus:1.1.1.2518.2” and the bottom is from “Locus:1.1.1.8343.2”

3.4.2 Horseradish peroxidase glycoprotein standard spectra

Glycoprotein stain (Pierce Glycoprotein Staining Kit, catalog number 24562) containing horseradish peroxidase (UNIPROT accession P00433[33]) was analyzed on an ABSciex Triple TOF 5600+, producing 1,891 fragmentation spectra (similarly to [34]).

The data were provided and processed blind without knowledge of its sample origins, only that sugars were present; like the 62 curated spectra, these sugars were not used in the analysis, only in the validation of the results. Thus, like the first data set, the only *a priori* expected masses are of four common sugar residues (Hexose, HexNAc, dHex, and NeuAc), as well as the neutron mass. It is important to note that the presence of amino acids was not expected.

Non-combinatorial results are shown with $d = 16$ for both the unweighted and weighted (Table 3.13) approaches.

The amino acids found with the $d = 16$ alphabet projection (*i.e.*, the alphabet reported in Table 3.14) are `G,T,I/L,N,` and `K/Q` (`K` and `Q` are listed together because the machine's ϵ is too large to differentiate between the two for the mass found). These amino acids can form a chain, `LNGNL`, which are the 241st through 245th amino acids in the peptide sequence. This includes the glycosylation site at the 244th amino acid (the second asparagine in `LNGNL`) in the sequence [33]. The amino acid chain `TLNTT` can also be produced from the alphabet. This chain covers the 226th through the 230th amino acids in the peptide sequence which includes another glycosylation site occurs at the 228th amino acid in the peptide sequence.

The weighted non-combinatorial approach, which found more amino acids than the unweighted non-combinatorial approach, was only able to find `I/L, T,` and `A`. Because of the lack of asparagine found by either non-combinatorial approach, neither one is able to build an amino acid chain which covers any of the glycosylation sites for this peptide.

Examples of recurring structures found using LSH with the $d = 16$ are shown in Figure 3.3.

Examples of two subspectra, from two different spectra, and their connected *de novo* graphs, which include the amino acid chain `LNGNL`, are shown in Figure 3.4.

Rank	Mass	Molecule
1	18.00000	Water
2	0.02500	
3	0.97500	Neutron/Deamidation
4	113.07500	I/L
5	203.07500	HexNAc
6	17.02500	Ammonia
7	17.00000	Ammonia
8	1.00000	Neutron/Deamidation
9	0.05000	
10	101.02500	T
11	0.00000	
12	18.02500	Water
13	17.97500	Water
14	27.97499	
15	113.05000	I/L
16	203.05000	HexNAc
18	162.05000	Hexose
54	146.05000	dHex
914	291.12500	NeuAc

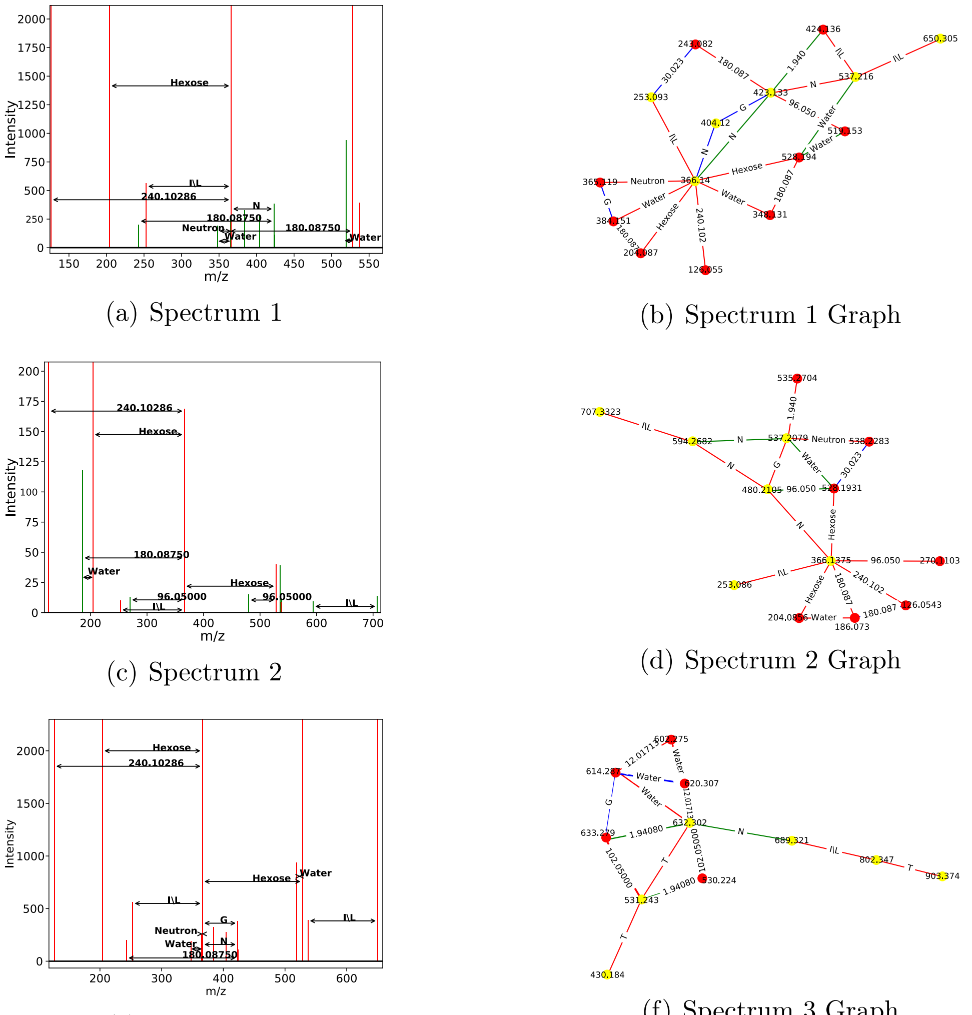
Rank	Mass	Molecule
1	18.00000	Water
2	0.02500	
3	203.07500	HexNac
4	113.07500	I/L
5	0.97500	Neutron/Deamidation
6	17.02500	Ammonia
7	0.00000	
8	17.00000	Ammonia
9	0.05000	
10	162.05000	Hexose
11	101.02500	T
12	35.99999	
13	1.00000	Neutron/Deamidation
14	203.05000	HexNac
15	41.02499	
16	17.97500	Water
53	146.05000	dHex
1,112	291.10500	NeuAc

Table 3.13 Most frequent $d = 16$ gap pairs ($i.e.$, $m_j - m_i$) on 1,891 glycoprotein standard spectra. The left table ranks using the unweighted frequency of gaps and the right table weights each gap by the product of peak intensities $p_i \cdot p_j$. Masses are rounded to 5 decimal points.

The combinatorial approach was run for 16 epochs per thread. Each epoch used 1000 iterations. The total real runtime of the analysis was 4 hours for $d = 16$ and 10.6 hours for $d = 64$. The acceptance rate eventually decays, and similar results may be achievable with lower runtimes.

Mass value	Manual interpretation	Known *a priori*?	Monoisotopic mass
1.02500	Neutron	Yes	1.00860
1.94080			
12.01713			
18.00000	Water	No	18.01056
30.02305			
42.02500			
57.00000	G	No	57.02146
96.05000			
101.04583	T	No	101.04767
102.05000			
113.06250	I/L	No	113.08406
114.05188	N	No	114.04292
128.06040	K/Q	No	128.09496/128.058578
162.06580	Hexose	Yes	162.04746
180.08750			
240.10286			

Table 3.14 Results when running the combinatorial approach on 1,891 glycoconjugate spectra with $d = 16$. Masses, they are reported in ascending order. Masses are rounded to 5 decimal points. Masses known *a priori* are labeled; these masses were not provided to the model, but instead are known true positives in advance.



(a) Spectrum 1

(b) Spectrum 1 Graph

(c) Spectrum 2

(d) Spectrum 2 Graph

(e) Spectrum 3

(f) Spectrum 3 Graph

Figure 3.4 **Subgraphs with an amino acid chain matching glycosylation sites.** Three spectra (a,c,e) and their corresponding *de novo* graphs (b,d,f) found using the combinatorial approach. The top two spectra contain the amino acid chain LNGNL and the bottom contains the amino acid chain TLNTT. Graphs use red edges to mark charge $z = 1$, green edges for $z = 2$, and blue edges for $z = 3$. The nodes colored in – yellow represent nodes touched by the amino acid chain. Figures (a,b) came from spectrum titled “Locus:1.1.1.8405.3” , figures (c,d) came from spectrum titled “Locus:1.1.1.8036.2”, and figures (e,f) came from “Locus:1.1.1.2523.2.”

3.5 Edge-maximal isomorphic subgraphs on embeddings in $\mathbb{Z}^d$

3.5.1 Solving edge-maximal isomorphic subgraph via cross-correlation

To test our methods we generated pairs of random graphs embedded in $\mathbb{Z}^2$. In Table 3.15 we compared the runtimes over pairs of graphs with similar amount of nodes. The brute force method iterated over all possible node bijections in the two graphs and counted the matching edges, which has exponential time complexity. The grid size is the maximum placement for a node; this is important because it controls the size of the adjacency tensor. The FFT approach is significantly faster and scales better for number of nodes while having 100% accuracy; however, the FFT is a divide and conquer algorithm which scales in speed and memory with the grid size (Table 3.16).

Embedding	$\lvert V_1\rvert$	$\lvert V_2\rvert$	Brute Force	FFT
32x32	5	8	0.10575	0.0052809
32x32	6	9	1.04943	0.0053590
32x32	7	10	11.8022	0.0053147
32x32	7	12	73.87356	0.0052950
32x32	7	13	147.53414	0.0052731

Table 3.15 Runtimes to find edge-maximal isomorphic subgraph matchings on graph pairs with different numbers of vertices, $|V_1|, |V_2|$. Brute force uses all possible node bijections, whereas FFT uses FFT-based cross-correlation on the $\mathbb{Z}^d$ embedding.

Embedding	Number Nodes	Time	Embedding	Number nodes	Time
32x32	100	0.06189	256x256	100	3.11040
64x64	100	0.16379	256x256	200	6.20085
128x128	100	0.69817	256x256	300	9.27673
256x256	100	3.12268	256x256	400	12.6026
512x512	100	18.8930	256x256	500	15.59189
1024x1024	100	102.573	256x256	600	18.4465
2048x2048	100	500.327	256x256	700	22.6432

Table 3.16 How FFT-based cross-correlation runtimes scale for finding edge-maximal isomorphic subgraphs between a pair of graphs with different embedding tensor dimensions and different numbers of nodes.

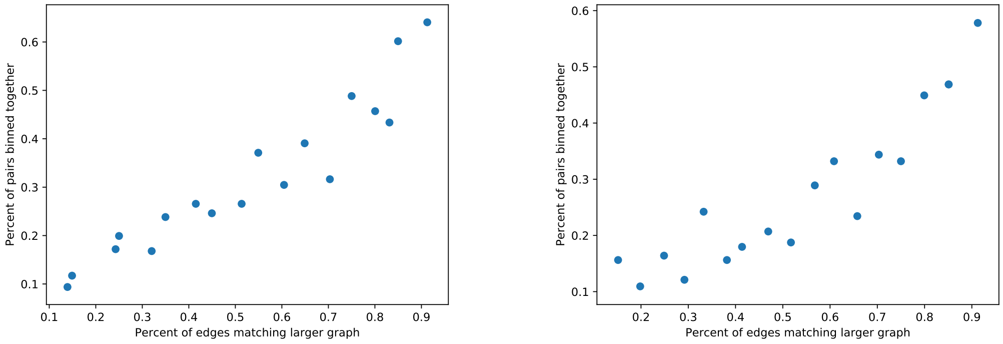


Figure 3.5 LSH matching quality for graphs embedded $\in \mathbb{Z}^2$ using 4-bit shift-invariant LSH. The left panel is run on proper supergraph–subgraph pairs, where the larger graph has 120 nodes and 812 edges. In this panel, subgraphs were found by randomly removing edges. The right panel compares, the right panel is run on graph pairs where both have 120 nodes and 812 edges. In this panel, the second graph was found by removing edges and then adding random new edges. LSH was performed 256 times to empirically estimate the probability that the graphs were binned together.

3.5.2 Finding graph pairs with highly isomorphic subgraphs via LSH

Figure 3.5 uses random graphs to demonstrate the effectiveness of the power spectrum as feature space for a shift-invariant LSH. Note that small changes to a graph (*e.g.*, removing an edge) can correspond to a non-trivial angle change between the two graphs in their power spectrum embeddings $\in \mathbb{R}^m$; thus, even similar graphs may be separated by a plane. For this reason, we can run a few full repetitions of LSH; the probability of a close match not colliding in at least one repetition will become small. The approach is trivially parallelizable; although we do not exploit that in this manuscript, it can mitigate the runtime penalty of performing replicate LSH runs.

Table 3.17 demonstrates that the LSH approach achieves a significant reduction in runtime while consistently finding graphs with large isomorphic subgraph matchings.

Avg. Nodes	Avg. edges	Hashes	Total	Match rank	FFT time	LSH time	Hash time
115	200	256	60	50	228.79458	47.97628	20.70615
143	300	256	47	29	291.55601	53.53428	26.15016
166	501	256	65	30	387.70664	86.29910	35.68782
206	401	256	42	28	372.05800	64.14751	33.16835
213	600	256	69	40	359.85256	82.97930	33.27186
158	500	512	151	90	335.60315	163.0908	60.66355
101	502	1024	78	57	333.87344	163.9954	110.6452

Table 3.17 Finding graph pairs with large edge intersection. 32 graphs (496 possible comparisons) were run using FFT between all pairs and using a 4-bit LSH hash. "Total" indicates the number of pairs that had non-trivial subgraph isomorphisms. "Match rank" indicates the number of these top graphs recovered by LSH (*e.g.*, 3 indicates that the largest 3 graphs were recovered by LSH). "LSH time" takes into account the time to calculate the hashes as well as the subsequent pairwise comparisons in each hash bin; "Hash time" reflects only the hashing step (which can be reduced significantly using parallelization), to underscore the ability to improve the LSH approach.

CHAPTER 4 DISCUSSION

4.1 Convex optimization

4.1.1 Minimal quadratically constrained linear program (M2.1.3)

From M2.1.3, this quadratically constrained linear program works well for samples that are much too small to be interesting. Unfortunately, the data do not become interesting until they are much too large to be run using this model.

4.1.2 Lagrangian relaxation (M2.1.4)

Before we applied Lagrangian relaxation, we attempted to relax the problem by allowing the edge indicator variables to be continuous in $[0, 1]$. If this worked, we would have a linear program with only continuous variables, which can be solved efficiently. Unfortunately, this can not work as a relaxed $e^{(\ell)}_{i,j,k}$ variable in constraint 2.3d is able to cheat the system. It was allowed to shrink as small as it could so that the middle term would be less than ϵ regardless of Δ_k. Then Δ_k was free to be any value.

This first Lagrangian relaxation 2.1.5 formulation was not much of an improvement over the quadratically constrained LP. This was to be expected as it is still a QP which is very difficult to solve.

The Lagrangian multipliers helped for some runs, but they often would not converge for many iterations, causing terrible runtimes. However, the solver would often get an alphabet with several correct masses (correct meaning there were two peaks whose mass difference was within ϵ of the alphabet mass) after the first iteration, but this alphabet would often change completely.

4.1.3 Minimizing over indicator variables (M2.1.5)

The second model to use Lagrangian relaxation (2.1.5) was created with the purpose of minimizing over the size of the alphabet and not the magnitude of the alphabet masses. This did have the intended effects because the alphabet when optimized over the indicator variables had larger values. Larger mass values are not necessarily desired, but there should not be a bias against them.

The major problem with the previous model was the speed and lack of convergence with the Lagrangian multipliers, neither of these were fixed with this new model.

4.1.4 An edge-centric model: incentivizing larger graphs (M2.1.6)

The third model to use Lagrangian relaxation was modified from the previous model simply by adding $C/(\sum_{i,j,k} e^{(\ell)}_{i,j,k})$ to the objective function. This certainly helped as this model turned on over twice as many edges and, importantly, incentivised more unique masses.

4.1.5 Removing the quadratic constraint

Using a trick for replacing a quadratic term in which one variable is a binary indicator variable and the other is continuous, we were able to make a problem with only linear constrained.

There are still binary edge indicator variables so our problem is a MILP. Also, most of the variables before removing the quadratic constraint were the edge indicator variables. There is one new $z^{(\ell)}_i, j, k$ continuous variable for each edge indicator variable, so there are almost twice as many variables as before. There are also four more constraints for each of the new $z_i^{(\ell)}, j, k$ variables. This significant increase in variables and constraints dampens the impact made by removing the quadratic constraint.

4.1.6 Maximize number of edges and finding unique masses through constraints (M2.1.9)

In model 2.1.9 we switched solvers from Mathematica to CPLEX. For this model the last constraint was added which dissallowed two alphabet masses to cover the same peak pair. This worked well as all alphabets masses are fully unique, so this model was a success in this aspect. Similar to previous models, the speed is still lacking. Also, minimizing over the indicator variables b_i was nullified by maximizing the amount of edge variables turned on and a correct value of γ which would allow some, but not all, masses was very hard to tune.

4.1.7 Two step method of finding the best Δs and then minimizing the alphabet (M2.1.10

This was by far the best model for a standard MILP implementation. Finding only one mass value at a time meant drastically reducing the amount of alphabet mass variables and edge indicator variables. These quantities are further reduced each iteration as peak pairs were taken away by previously found masses.

We want to optimize over hundreds of peaks across hundreds or thousands of spectra. Despite the massive speed-up over previous models, the model was still not fast enough. Since the first pass optimization was never fast enough, the second pass was never implemented.

4.1.8 Maximum vertex cover approach (M3.2.7)

The bipartite graph optimization model (3.2.7) was by far the best convex optimization model. It was fast enough to find an alphabet of 8 masses on the glycoconjugate dataset which consists of 62 spectra with different charges 1,2, and 3 in less than half an hour.

The model was left too unconstrained to find a great alphabet. For example, one alphabet mass can be a multiple of another which leads to masses like 23.8847 and 35.91240 which are very close to twice carbon (12.00Da) and twice water (18.01057). In the future we may revisit this model to see if we can efficiently encode some further constraints such as no redundant masses or disallowing one mass to be the sum of two others, etc.

Despite clever methods such as branch-and-bound, binary linear programs are still NP-hard and become a nightmare for large problem sizes, for this reason we abandoned this model as well.

We attempted to relax the bipartite graph optimization by letting the binary variables be continuous variables between 0 and 1. This ILP has a linear objective function and linear constraints so a true LP version could potentially be very fast. However, we were not able to get the edge indicator variables to be pushed into $\{0, 1\}$. This is because constraints 2.8h and 2.8i want to keep many edge indicators at at low, but non-zero, value.

4.2 A max-flow/min-cut formulation

Though we did not find any great results for the graph-cut models, we still hold out hope. Intuitively, they feel like a good fit. We can weight edges between Δ nodes and the USED and NOT USED nodes by how well the alphabet mass connects peaks in the data and weight edges between Δ nodes based on how many pairs of adjacent edges they share. Then use an efficient min-cut algorithm to solve the problem. Unfortunately, it has been hard finding the right model.

For earlier models, it became apparent that in order for the model to work, the α and β values could not be uniform across the model. This caused the models output to be based entirely on which is larger: α times the number of all possible masses or the sum of all edge weights $p_i p_j$. If the former is larger, then the alphabet would be empty. If the latter is larger, then the alphabet consist of all possible masses.

In the future we hope to work more on these graph-cut models. None of these models may be correct, but we believe graph-cuts should somehow solve the minimum alphabet problem. One forseeable problem is having datasets which can form millions of possible alphabet masses. In this case, it may be best to solve each spectra separately then adjust all spectra's edge weights based on shared alphabet mass values. After many iterations the edge weights may converge to a well defined alphabet.

4.3 Markov chain Monte Carlo

4.3.1 An alphabet for 62 expert-curated spectra including neutron, water, sugars, and more

Even though no masses or chemical knowledge were provided to the combinatorial approach and we only expected four sugar resides and the neutron mass in advance, our approach finds masses close to water and ammonia on the 62 expert-curated spectra. The mass we do find that is within ϵ of the mass of a Neutron is also within ϵ of the mass difference caused by deamidation. Deamidation is a modification to amino acids where a nitrogen and a hydrogen are replaced by an oxygen with a mass difference of 0.984Da. These are both plausible, particularly since this data came from a urine sample. We also find masses close to Hexose and to HexNAc in this data. While the non-combinatorial approach does not assign a high rank to HexNAc, the combinatorial approach finds it with $d = 8$ because the connectivity improvement of HexNAc is superior enough to justify its low frequency and incidence to low-intensity peaks. Interestingly, we also find a mass at 88.01555Da. This matches the difference between several pairs of saccharide oxonium ions [36]: Neu5Ac (292.103Da) - HexNAc$^+$ (204.087Da) = 88.0162Da, [Neu5Ac-H_2O]$^+$ (274.092Da) - [HexNAc-H_2O]$^+$ (186.076Da) = 88.0159Da. Those are instances where the alphabet mass connects two whole glycan oxonium ions, but it also connects [HexNAc-2H_2O]$^+$ (168.066Da) to 256.082Da and [HexNAc - $C_2H_4O_2$]$^+$ to 232.081Da. It appears that Neu5Ac generates a series of oxonium ions 292.103Da, 274.092Da, 256.082Da, and 232.081Da. The second and third result from the loss of a water molecule and the last results from the loss of two carbons. HexNAc generates series of oxonium ions 204.087Da, 186.076Da, 168.066Da, and 144.065Da. Similar to Neu5Ac the first two mass shifts are due to the loss of water molecules and the final shift is due to the loss of two carbon.

The other two unknown masses are 30.01058Da and 42.01071Da. 30.01058Da is very close to the isotopic mass of H_2CO, 30.010565Da. There are a few different things which can create a neutral loss of H_2CO: the molecule hydroxymethyl, a serine and glycine substitution, a glycine and serine substitution, or a formaldehyde induced modification[35].

Similar to 30.010565Da, there are a few known modifications which could create the 42.01071Da neutral loss: a glutamic acid and serine substitution or acetylation [35]. Similar to 88.01555Da, there may be other analytes or differences between two other mass changes which form 30.010565Da and 42.01071Da.

4.3.2 An alphabet for 1,891 glycoprotein standard spectra including neutron, amino acids, sugars, and more

On the 1,891 glycoprotein standard spectra, our approach discovers multiple amino acid masses without prior knowledge that are in the samples contained peptides. For $d = 16$ the combinatorial approach found glycine, arginine, and one or both of Lysine/Glutamine when neither non-combinatorial approach did. However, the weighted non-combinatorial approach found alanine which the combinatorial approach did not. Both the combinatorial and non-combinatorial approaches found isoleucine/leucine and threonine.

the fact that the combinatorial approach finds glycine and arginine is important because the amino acids in the alphabet can form the chains *LNGNL* and *TLNTT*. *LNGNL* covers the 241^{st} through 245^{th} amino acids in the peptide sequence which includes the glycosylation site at the 244^{th} amino acid (the second asparagine in *LNGNL*) in the sequence [33]. Similarly, *TLNTT* covers the 226^{th} through the 230^{th} amino acids in the peptide sequence which includes another glycosylation site occurs at the 228^{th} amino acid in the peptide sequence.

Both of the 30.02305Da and 42.02500 mass differences are within ϵ of the mass differences discussed in the previous section so all possible explanations of those mass differences apply here as well. Similar to the alphabet for the 62 expert-curated spectra, the mass found which is within ϵ of a neutron mass is also within ϵ of deamidation.

4.3.3 Future improvements

Possible improvements to the model include parameterizing a penalty on masses too close to one another or even triplets of masses where $\Delta_1 \approx \Delta_2 + \Delta_3$. The user could supply a list of peaks in which the program should favor or be forced to connect such as a precursor peak.

Since the method allows for us to seed the initial masses from the combinatorial approach, there will probably be benefit to seeding them with the results of the non-combinatorial approach or to seeding them with available prior knowledge (*i.e.*, the neutron mass and the four sugar residues) or with any masses known to be in the sample *a priori*.

Neither dataset was charge deconvolved. However, charge deconvolution would allow the graph building method to only connect peaks by $\frac{\Delta_k}{z}$ when the two peaks have charge equal to z.

An approach to making our method semi-supervised could be as follows: First, run the program as it currently is to get an original alphabet. Second, try and find a known molecule in the alphabet (*i.e.* through mass decomposition) and populate a new alphabet with a family of molecules based on this known molecule. For example, if you blindly find an amino acid, then rerun the program with an alphabet larger than 21, seeding the first 21 with the amino acid masses (use "-f" flag to protect the seeded alphabet masses). Similarly, if you blindly find a sugar, then rerun the program while seeding the alphabet with sugar masses. This could be particularly useful for finding something like a post-translational modification on a peptide once an original alphabet containing amino acids is found.

4.3.4 Recurring subgraphs

By finding an alphabet Δ and subgraphs that have a high degree of isomorphism to one another (Figure 3.2 and Figure 3.4), we find results consistent with standard sugar trees [7]. Because we expect a good alphabet Δ to produce connected components from different spectra with large isomorphic subgraphs, then it may be possible to invert this notion: by first clustering spectra that have similar peaks (up to mass shifts), then we could possibly use those clustered spectra to help estimate the alphabet Δ.

The convolutional/LSH approach proposed here may also be used to find spectra containing graphs with graph products [9]. This may be useful for inferring chemical structure from the graphs built in this paper.

4.4 Edge-maximal isomorphic subgraphs on embeddings in $\mathbb{Z}^d$

When considering a specific embedding pair, the FFT-convolution-based approach for finding edge-maximal isomorphic subgraphs is superior to the brute force method even for very small graphs and scales much better.

When considering several pairs, the results of 256 4-bit hashes shows the LSH discriminates among graphs. It is more likely to bin graphs with larger edge-maximal isomorphic node bijections than smaller. In this manner, the shift-invariant LSH approach reduces the runtime while still finding a significant quantity of the graphs with large subgraph isomorphisms.

Figure 3.5 demonstrates that the hashing function is effective at finding similar graphs together; however, the probability of two graphs being split by a plane given the number (or percentage) of isomorphic edges similarity has only been observed empirically and has not been derived and could prove useful for improving the hashing method. A theoretical bound on this probability could help develop further embeddings for graphs that accomplish superior LSH discrimination in fewer hashes.

We use the power spectrum in order to bin the graphs, but this has a few flaws. When an FFT is converted to a power spectrum, the phase information is lost. For this reason, it is possible that two different graphs may produce the same power spectrum. An approach that uses a different feature space for LSH or that keeps this discarded phase information in a shift-invariant manner would eliminate these spurious collisions. One option that would also reduce the need for a dense embedding would be to store the graph in a fully sparse manner and then compute several random frequency components of the Fourier transform; different graphs that coincidentally produce the same power spectrum may not overlap at other frequencies not visited by the FFT.

Our shift-invariant LSH method assumes the graphs are in the proper orientations relative to each other to result in the maximal isomorphic subgraph matching. There may be some way we can overcome this assumption, one possible avenue may be to project onto a unit sphere, where discretization over angles would permit rotation instead of shifting. One-dimensional embeddings only have two possible orientations (left-to-right and right-to-left), and so the approach proposed here already solves the oriented problem with $d = 1$.

While everything here has been on the discussion of Euclidean graphs embedded in $\mathbb{Z}^d$ it is possible to use similar methods on graphs embedded in $\mathbb{R}^d$ by performing naive discrete d–dimensional Fourier transform on fully sparse adjacency tensor rather than dense d-dimensional FFT. Alternatively you could bin the graphs and use as vectors; this has been done successfully on *de novo* graphs from mass spectra, which can be embedded in $\mathbb{R}^1$[37].

CHAPTER 5 SOURCE CODE AVAILABILITY

Links to resources for source code will attempt to be kept stable at `https://patkreitzberg.com`. Specific source code repositories are mentioned below:

Mass spectrometry alphabet projection

The source code and data used for Markov chain Monte Carlo methods can be found at `https://bitbucket.org/orserang/peak-bagger` or `https://PatKreitzberg@bitbucket.org/PatKreitzberg/peak-bagger.git`. This includes `C++` code for both the non-combinatorial and combinatorial approaches, `python` scripts for plotting and annotating spectra, and `python` scripts for performing LSH hashing to find recurring subgraphs in the spectra. Code and data are provided with an MIT license.

Edge-maximal isomorphic subgraphs on embeddings in $\mathbb{Z}^d$

Source code (MIT license) is available here: `https://PatKreitzberg@bitbucket.org/PatKreitzberg/edge-maximal-subgraph-isomorphism.git`.

BIBLIOGRAPHY

[1] G.W. Hart and R. J. Copeland. Glycomics hits the big time. *Cell*, 143(5):672–676, 2010.

[2] Monya Baker. Metabolomics: from small molecules to big ideas, 2011.

[3] K. Dührkop, M. Fleischauer, Marcus Ludwig, A. V. Melnik, M. Meusel, P. C. Dorrestein, J. Rousu, and S. Böcker. Sirius 4: a rapid tool for turning tandem mass spectra into metabolite structure information. *Nature Methods*, 16(4):299–302, 2019.

[4] S.P. Cleary, A.M. Thompson, and J.S. Prell. Fourier analysis method for analyzing highly congested mass spectra of ion populations with repeated subunits. 88(12):6205–6213, 2016.

[5] A. Frank and P. Pevzner. Pepnovo: de novo peptide sequencing via probabilistic network modeling. *Analytical Chemistry*, 77:964–973, 2005.

[6] K. Medzihradszky and R. Chalkley. Lessons in *de novo* peptide sequencing by tandem mass spectrometry. *Mass Spectrom Review*, 34(1):43–63, 2015.

[7] O. Serang, J. W. Froehlich, J. Muntel, G. McDowell, H. Steen, R. S. Lee, and J. A. Steen. SweetSEQer, simple de novo filtering and annotation of glycoconjugate mass spectra. *Molecular & Cellular Proteomics*, 12(6):1735–1740, 2013.

[8] P. Hong, H. Sun, L. Sha, Y. Pu, K. Khatri, X. Yu, Y. Tang, and C. Lin. GlycoDeNovo - an efficient algorithm for accurate *de novo* glycan topology reconstruction from tandem mass spectra. *Journal of the American Society for Mass Spectrometry*, 28(11):2288–2301, 2017.

[9] S. Bhatia, Y. J. Kil, B. Ueberheide, B. T. Chait, L. Tayo, L. Cruz, B. Lu, J. R. Yates III, and M. Bern. Constrained *de novo* sequencing of conotoxins. *Journal of Proteome Research*, 11(8):4191–4200, 2012.

[10] A. Guthals, J. D. Watrous, P. C. Dorrestein, and N. Bandeira. The spectral networks paradigm in high throughput mass spectrometry. *Molecular bioSystems*, 8(10):2535–2544, 2012.

[11] E. Benedetti, M. Pučić-Baković, T. Keser, A. Wahl, A. Hassinen, J. Yang, L. Liu, I. Trbojević-Akmačić, G. Razdorov, J. Štambuk, L Klarić, I. Ugrina, M. Selman, W. Manfred, I. Rudan, O. Polasek, C. Hayward, H. Grallert, K. Strauch, A. Peters, T. Meitinger, C. Gieger, M. Vilaj, Boons G., K. Moremen, T. Ovchinnikova, N. Bovin, F. Kellokumpu, S. Theis, G. Lauc, and J. Krumsiek. Network inference from glycoproteomics data reveals new reactions in the igg glycosylation pathway. *Nature communications*, 8(1):1483, 2017.

[12] B.A. Budnik, J. V. Olsen, T. A. Egorov, V. E. Anisimova, T. G. Galkina, A. K. Musolyamov, E. V. Grishin, and R. A. Zubarev. *De novo* sequencing of antimicrobial peptides isolated from the venom glands of the wolf spider lycosa singoriensis. *Journal of Mass Spectrometry*, 39(2):193–201, 2004.

[13] Shubhra Rastogi, S. Meena, A. Bhattacharya, S. Ghosh, R. Kumar Shukla, N. Singh Sangwan, R. Kishori Lal, M. Mohan Gupta, U. Chandra Lavania, V. Gupta, D.A. Nagegowda, and A. Kumar Shasany. De novo sequencing and

comparative analysis of holy and sweet basil transcriptomes. *BMC Genomics*, 15(1), 2014.

[14] M. E. Graham, M. Thaysen-Andersen, N. Bache, G. E. Craft, M. R. Larsen, N. H. Packer, and P. J. Robinson. A novel post-translational modification in nerve terminals: O-linked n-acetylglucosamine phosphorylation. *Journal of Proteome Research*, 10(6):2725–2733, 2011.

[15] V. Dancik, T.A. Addona, K.R. Clauser, J.E. Vath, and P.A. Pevzner. *De novo* peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 6(3-4):327–342, 1999.

[16] N. Bandeira. Spectral networks: a new approach to de novo discovery of protein sequences and posttranslational modifications. *BioTechniques*, 42(6), 2018.

[17] B. Greenhill, S. Valtier, and J.T. Cody. Metabolic profile of amphetamine and methamphetamine following administration of the drug famprofazone. *Journal of Analytical Toxicology*, 27(7):479–484, 2003.

[18] GP Reynolds, JD Elsworth, K. Blau, M. Sandler, AJ Lees, and GM Stern. Deprenyl is metabolized to methamphetamine and amphetamine in man. *British Journal of Clinical Pharmacology*, 6(6), 1978.

[19] A. Schrijver. *Theory of Linear and Integer Programming*, page 83. Wiley-Interscience, 1986.

[20] s. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974.

[21] P. Rubin. Binary variables and quadratic terms. https://orinanobworld.blogspot.com/2010/10/binary-variables-and-quadratic-terms.html, 2010.

[22] L.R. Ford Jr. and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[23] M. Geomans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):115–1145, 1995.

[24] M. Cygan, M. Mucha, K. Węgrzycki, and M. Włodarczyk. On problems equivalent to (min,+)-convolution. *arXiv preprint arXiv:1702.07669*, 2017.

[25] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[26] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[27] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[28] M. Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2, 1986.

[29] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 618–625. ACM, 1997.

[30] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[31] L. Wang, S. Li, and H. Tang. msCRUSH: fast tandem mass spectral clustering using locality sensitive hashing. *Journal of Proteome Research*, 18(1):147–158, 2018.

[32] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[33] The UniProt Consortium. Uniprot: a worldwide hub of protein knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, 2019.

[34] C. Zhang, Z. Ye, P. Xue, Q. Shu, Y. Zhou, Y. Ji, Y. Fu, J. Wang, and F. Yang. Evaluation of Different N-Glycopeptide Enrichment Methods for N-Glycosylation Sites Mapping in Mouse Brain. *Journal of Proteome Research*, 15(9):2960–2968, 2016.

[35] D.M Creasy and J.S. Cottrell. Unimod: Protein modifications for mass spectrometry. *Proteomics*, (4):1534–1536, 2004.

[36] A. Halim, U. Westerlind, C. Pett, M. Schorlemer, U.J. Rüetschi, G. Brinkmalm, C. Sihlbom, J. Lengqvist, G. Larson, and J. Nilsson. Assignment of saccharide identities through analysis of oxonium ion fragmentation profiles in lc-ms/ms of glycopeptides. *Journal of Proteome Research*, 13 12:6024–32, 2014.

[37] P. Kreitzberg, M. Bern, S. Qingbo, F. Yang, and O. Serang. The alphabet projection of spectra. Submitted.