Graphical Models for Peptide Identification of Tandem Mass Spectra

John T. Halloran

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Jeff A. Bilmes, Chair

William S. Noble, Chair

Katrin Kirchoff

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Graphical Models for Peptide Identification of Tandem Mass Spectra

John T. Halloran

Co-Chairs of the Supervisory Committee:
Professor Jeff A. Bilmes
Electrical Engineering

Professor William S. Noble
Genome Sciences

Graphical models (GMs) provide a flexible framework for modeling phenomena. In the past few decades, GMs have become indispensable tools for machine learning and computational biology. They afford a wide range of modeling granularity, from restricting only exact events of interest to occur to allowing all possible events in a phenomenon's event space. For all such modeling considerations, GMs afford efficient algorithms to perform inference over the probabilistic quantities of interest. In this thesis, we show how GMs may be leveraged to improve both identification accuracy and search runtime of tandem mass (MS/MS) spectra. For the majority of existing MS/MS scoring algorithms, we give equivalent GMs and show how search time may be algorithmically improved. We present GMs for posterior based (sum-product) and max-product inference which offer state-of-the-art performance and, most importantly, are amenable to efficient parameter estimation. Furthermore, we show how a GM which generatively models the stochastic process by which peptides produce MS/MS spectra may be utilized to calculate features for improved classification between correctly and incorrectly identified spectra, leading to significantly improved identification accuracy.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

People often say it is about the journey, not about the destination. I've watched others complete their PhDs with seeming complete ease and amazing grace, much in contrast to myself. Nonetheless, while no one sets out to make tasks more difficult than they need be, I am happy with my journey and the test of self; I now know who I am. As the great Will Ferrell once said, "before you marry a person you should first make them use a computer with slow internet to see who they really are." As my long academic journey draws to a close, I am filled with both profound happiness and deep sadness. In the heart of this dichotomy of emotions, there is a calm; I am deeply satisfied and proud of all we (my advisors, lab cohorts, and self) have accomplished.

Firstly, I would like to thank my advisor, Jeff Bilmes, for all the support and guidance over the years. Thank you for being a mentor over the years and for teaching me so much about graphical models and machine learning. Thank you for working tirelessly to ensure that I and my fellow MELODI lab members had ample resources to get our work done, including securing the lab itself and building an environment for our ideas to flourish. Above all else, thank you for your understanding, patience, and friendship. I look forward to sharing many sour beers in the future. I would also like to thank my advisor, Bill Noble, for the great amount of support over the years. Thank you for all your guidance, particularly in tandem mass spectrometry, of which I knew nothing about prior to our working together but on which I am now, suffice to say, hooked. Perhaps it goes without saying, but thank you for being entirely welcoming when it came to my being part of the Department of Genome Sciences, the noblelab, and the mass spectrometry community. With that, I would also like to thank the University of Washington Department of Genome Sciences and the NIH for funding me

through the Genome Training Grant.

I would like to offer thanks to my committee members, Katrin Kirchoff and Ali Shojiae, for all their time, patience, and support. Also, sorry for the relentless onslaught of emails.

I would like to thank all of my fellow MELODI labmates for the long talks, discussions, advice, and general encouragement over the years. In no particular order, I extend my sincerest gratitude to Rishabh Iyer, Kai Wei, Shengjie Wang, Chandrashekhar Lavania, Wenruo Bai, and Jennifer Gillenwater. May we remain thick as thieves even after life pulls us far apart from the walls of the MELODI lab. I will infinitely miss our time together.

Thank you to all my collaborators over the years, who include Shengjie Wang, Wenruo Bai, Jie Liu, and Karthik Mohan. You've all taught me that collaboration is truly the key to making research interesting and fruitful. Most importantly, you've all taught me collaboration is the most enjoyable part of research.

A special thanks goes out to Richard Rogers. Whenever I broke things in GMTK, you were quick to come to my made aid and unbreak things.

I would like to offer special thanks to John Weisenberger, Yuichi Sugawara, and Cale Regidor. You've all given me a shoulder to lean on whenever times have been tough. I hope I can return the favor, I actually have two shoulders for those extra tough times.

I would like to take this opportunity to thank my undergraduate mentors at Seattle University. In no particular order, thank you Professors Chris Black, John Carter, Agnieszka Miguel, Margarita Takach, and Donna Sylvester. Without your encouragement and help, I never would have made it into graduate school, much less obtained my doctorate.

Last but certainly not least, I would like to thank my wonderful wife Katrina Halloran, without whom this dissertation would not exist and I certainly never would have been a graduate student or even an engineer. In the face of all the good times and bad, Katrina and Bella have acted as a pillar of support upon which I could endure.

Stay classy Seattle, I'll miss you.

DEDICATION

to Katrina and Bella, to friends and loved ones who passed: Janice Halloran, Daniel Strickland, and Stefvin Caswell.

Chapter 1

INTRODUCTION

A fundamental problem in biology and medicine is accurately identifying the proteins present in a complex sample, such as a drop of blood. The solution to this problem has many important applications, such as the early detection of diseases and congenital defects. For instance, sickle-cell disease is caused by a single substitution in the amino acid sequence of a protein found in humans [57], and the identification of this mutated protein can lead to preventive care [71]. The only high-throughput method for solving this problem is *tandem mass spectrometry* (*MS/MS*), accomplishing in seconds what previously took hours using methods such as Edman's degradation [68].

Given a complex sample, an MS/MS experiment produces a collection of spectra typically on the order of thousands, each of which represents a single peptide (protein subsequence) that was present in the original sample. Fundamental to MS/MS is the ability to accurately identify the peptide responsible for generating a particular spectrum, which we call the *spectrum identification problem*. The most accurate methods for identifying MS/MS spectra make use of a peptide database [68], typically derived from the mapped genome of the organism of interest.

Given a peptide drawn from the database and an observed spectrum, database search methods compare a *theoretical spectrum* of the peptide's idealized fragmentation events to a preprocessed observed spectrum. The spectrum identification problem is greatly complicated by experimental noise, corresponding both to the presence of unexpected peaks (*insertions*) and the absence of expected peaks (*deletions*) in the observed spectrum (displayed in Fig. 2.3).

In this work, we show the many ways *graphical models* may be utilized for the spectrum identification problem. In the past few decades, graphical models have become a staple tool

of machine learning and bioinformatics. For instance, the *hidden Markov model* has been used to great success to model time series such as speech and genomics data in order to perform *inference*, i.e., in order to answer queries of interest with regards to the modeled data. Similarly, the models we use to solve the spectrum identification problem cast the problem as inference in a time series, where "time" for some of these models is defined in terms of the sequence of theoretical peaks and time for other models is defined in terms of the sequence of observed spectrum peaks.

Much of this thesis focuses on two graphical models introduced for the sole purpose of identifying tandem mass spectra (Chapter 6). In the first such model, Didea, the time series being modeled is the sequence of characters in a peptide string (called *amino acids*). In the second such model, DRIP, the time series being modeled is the sequence of observed spectrum peaks, the theoretical spectrum is hidden, and insertions and deletions are explicitly modeled. DRIP may be thought of as aligning the theoretical and observed spectra in a manner similar to edit distance (i.e., string alignment). Importantly, both DRIP and Didea allow efficient parameter estimation, where the learned parameters lead to improved search accuracy (Chapter 8). This is in stark contrast to many existing scoring algorithms, which do not support learning parameters but rather rely on hardcoded parameter values, often selected by hand-tuning.

As we can already see based on their brief descriptions, DRIP and Didea differ significantly from one another (in fact, their notions of time are inverted with respect to one another), perhaps the most striking difference between DRIP and Didea being the use of the max-product algorithm and the sum-product algorithm, respectively, to perform inference. These two algorithms are the most common two instances of *belief propagation* (an umbrella of algorithms for performing inference in graphical models, further discussed in Chapter 5.1) and highlight two important distinctions between the two models: while both DRIP and Didea identify tandem mass spectra well in practice, the most probable sequence of insertions and deletions is calculated in DRIP to score peptides. As such, performing inference in DRIP offers valuable information regarding a peptide given an observed spectrum. We utilize this

auxiliary information to extract features which may be used to further improve identification accuracy (Chapter 9).

While utilizing graphical models to improve search accuracy is a common thread in this work, we also discuss how approximate inference in graphical models may be utilized to algorithmically speed up identification of MS/MS data (Chapter 7). Importantly, we show how many commonly used search algorithms (such as SEQUEST [20], X!Tandem [14], Morpheus [73], and the base scores of MS-GF+ [44] and OMSSA [24]) may be represented as graphical models and thus may also be algorithmically sped up. We provide timing results for DRIP and SEQUEST's XCorr, which provide up to a ten-fold speed improvement for both low-resolution and high-resolution tandem mass spectra.

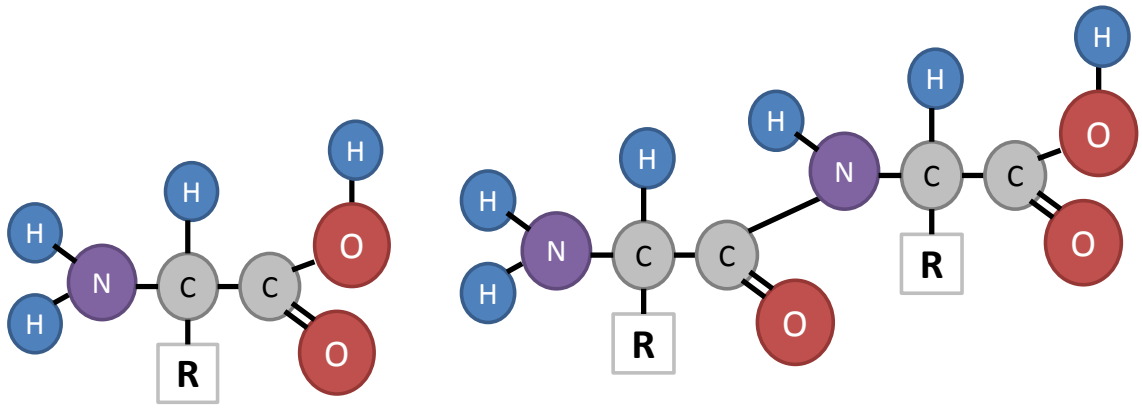
Chapter 2

TANDEM MASS SPECTROMETRY

Proteins are strings of characters called *amino acids*. There are twenty amino acids, each of which is identical save for their side-chains, depicted as the molecular group R in Figure 2.1(a). The far left group H_2N (called the *amino group*) denotes the beginning of the amino acid, often referred to as the *N-terminus*. The far right group COOH (called the *carboxyl group*) denotes the end of the amino acid and is often referred to as the *C-terminus*. A substring of a protein is called a *peptide* or *polypeptide*, which consists of two or more amino acids and which is typically, in practice, an order of magnitude shorter than its parent protein.

Although we are typically interested in the protein content of a complex mixture, the fundamental unit of observation in tandem mass spectrometry is the peptide, because peptides are more amenable to liquid chromatography and mass spectrometry analysis. Thus, a typical MS/MS experiment (Figure 2.2) begins by digesting the proteins into peptides using a cleavage agent, such as the enzyme trypsin. The peptides are separately loaded into the mass spectrometer via a process called liquid chromotography, wherein the digested proteins are loaded into a thin glass column containing a solvent. The solvent separates peptides based on chemical properties, such as hydrophobicity, and the peptides elute down the column at different speeds. The peptides exit out of a thin tip at the end of the column, are vaporized and ionized, and enter the mass spectrometer for the first round of mass spectrometry.

The first round of mass spectrometry, the *MS1* phase, measures the mass-to-charge ratio (m/z) of the intact peptide (called the *precursor* m/z). The intact peptide precursor ions selected in the MS1 phase undergo a second round of mass spectrometry, typically called the *MS2* phase, wherein they are fragmented using one of various techniques (such as



(a) Amino acid molecular structure where R is the side-chain group.

(b) Dipeptide molecular structure.

Figure 2.1: Amino acid and dipeptide. A dipeptide formed by a peptide bond between two amino acids, during which process an OH is lost from the left amino acid's C-terminus and an H is lost from the right amino acid's N-terminus (i.e., a water molecule is lost in total).

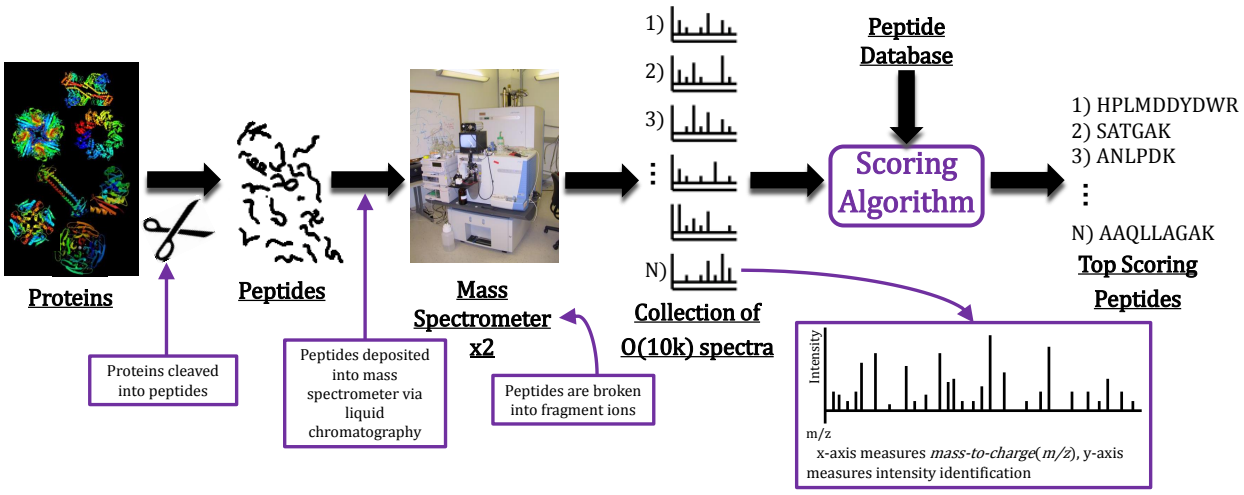


Figure 2.2: A typical MS/MS experiment.

collision-induced dissociation, the most commonly technique for peptide fragmentation) and the m/z of the values of the resulting fragments are measured. Each of these fragment m/z values is associated with an intensity value, which is roughly proportional to the number of copies of that peptide fragment. Figure 2.3 displays a sample tandem mass spectrum, along with the theoretical fragment ions (described below) of the generating peptide. A single unit along the m/z axis is called a *Thomson* (Th), and the intensity y-axis is unitless but can be seen as a measure of abundance or count.

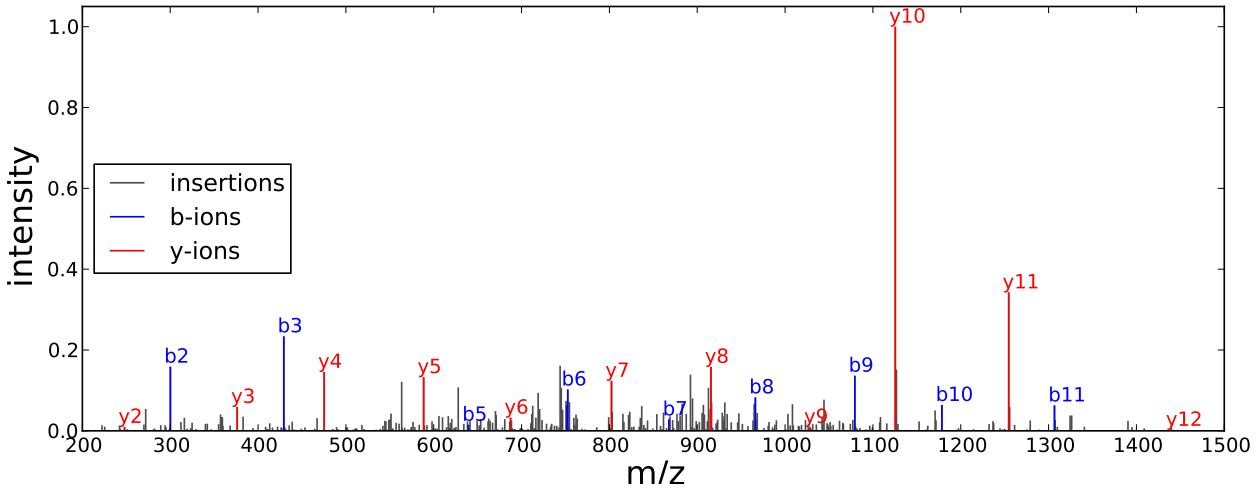


Figure 2.3: Real tandem mass spectrum, where the peptide responsible for generating the spectrum is x = LWEPLLDVLVQTK, the precursor charge c^s is 2. The b-ion peaks are colored blue, y-ion peaks are colored red, spurious peaks are colored gray.

2.1 Database search

Let $\mathbb{P}$ be the set of all peptides and S be the set of all tandem mass spectra. Given an observed spectrum $s \in S$ with observed precursor m/z m^s and precursor charge c^s, our task is to identify the peptide x from a given peptide database $\mathcal{D} \subseteq \mathbb{P}$ that is responsible for generating s. Any given mass spectrometry device is capable of isolating peptides with a specified precision on the precursor m/z; therefore, we may constrain the search to only

consider peptides with precursor m/z $\pm w$ of m^s. The set of *candidate peptides* to be scored is then

$$D(m^s, c^s, \mathcal{D}, w) = \left\{ x : x \in \mathcal{D}, \left| \frac{m(x)}{c^s} - m^s \right| \leq w \right\}, \tag{2.1}$$

where $m(x)$ is the calculated mass of peptide x. The goal of database search, then, is to return the highest scoring candidate peptide

$$x^* = \underset{x \in D(m^s, c^s, \mathcal{D}, w)}{\operatorname{argmax}} \psi(x, s), \tag{2.2}$$

where $\psi : \mathbb{P} \times S \to \mathbb{R}$ is a function that assigns higher scores to higher quality matches and the pair (x, S) is referred to as a *peptide-spectrum match* (PSM). The primary distinguishing characteristic of any database search procedure is its choice of score function ψ.

2.2 Theoretical spectra

Many scoring functions work by comparing the observed spectrum to a *theoretical spectrum* that is derived from a candidate peptide using basic rules of biochemistry. Let $x \in D(m^s, c^s, \mathcal{D}, w)$ be an arbitrary candidate peptide of length n and $\tilde{n} = n - 1$. Note that $x = x_0 x_1 \ldots x_{n-1}$ is a string of *amino acids*, i.e., characters in a dictionary of size 20. We use the notation $0 : k$ to represent the set of integers $\{0, 1, \ldots, k\}$ and, for a sequence such as x and for $k \leq n - 1$, $x_{0:k}$ to denote the sequence $x_{0:k} = x_0 \ldots x_k$. Our goal is to produce a theoretical spectrum v^x containing the fragment m/z values that we expect x to produce. We assume that the mass spectrometer employs collision-induced dissociation (CID), which is the most widely employed method of peptide fragmentation. The model can be modified in a straightforward fashion to accommodate other fragmentation modes.

The first type of fragment m/z value corresponds to prefixes or suffixes of the candidate peptide, referred to respectively as *b-ions* and *y-ions*. These b- and y-ions are functions $b(\cdot, \cdot, \cdot)$ and $y(\cdot, \cdot, \cdot)$, respectively, defined as:

$$b(x, c^b, k) = \frac{\sum_{i=0}^{k-1} m(x_i) + c^b}{c^b} \qquad y(x, c^y, k) = \frac{\sum_{i=\tilde{n}-k}^{\tilde{n}} m(x_i) + 18 + c^y}{c^y}. \tag{2.3}$$

where c^b and c^y are charges related to the precursor charge state. Note that the whole peptide mass is not considered. The b-ion offset corresponds to the mass of a c^b charged hydrogen atom, while the y-ion offset corresponds to the mass of a water molecule (18) plus a c^y charged hydrogen atom. When $c^s \in \{1, 2\}$, c^b and c^y are unity because, for any b- and y-ion pair, an ion with no charge is undetectable in the mass spectrometer and it is unlikely for both charges in a +2 charged precursor ion to end up on a single fragment ion [45]. When $c^s \geq 3$, we search both singly and doubly charged ions so that $c^b, c^y \in \{1, 2\}$.

As is common in the field, in the event that $c^s = 2$, when there is also no ambiguity as to the peptide being described, we also represent the b- and y-ion pairs as (b_k, y_{n-k}) for $k = 1, \ldots, \tilde{n}$, where the subscript denotes the number of amino acids utilized in the ion computation. As an example, for peptide x = EALK and precursor charge $c^s = 2$, $(b_1, y_3) = (b(\text{E}, 1, 1), y(\text{ALK}, 1, 1)) = (130, 331)$, where we've integerized the m/z values. Denoting the number of unique b- and y-ions as n^x and letting $\widetilde{n}^x = n^x - 1$, our theoretical spectrum is thus a sorted vector $v^x = (v_0, \ldots, v_{\widetilde{n}^x})$ consisting of the unique b- and y-ions of x. Figure 2.4 displays the theoretical spectrum for $x = \text{EALK}, c(x) = 2$, and Figure 2.3 displays an observed spectrum with annotated b- and y-ions.

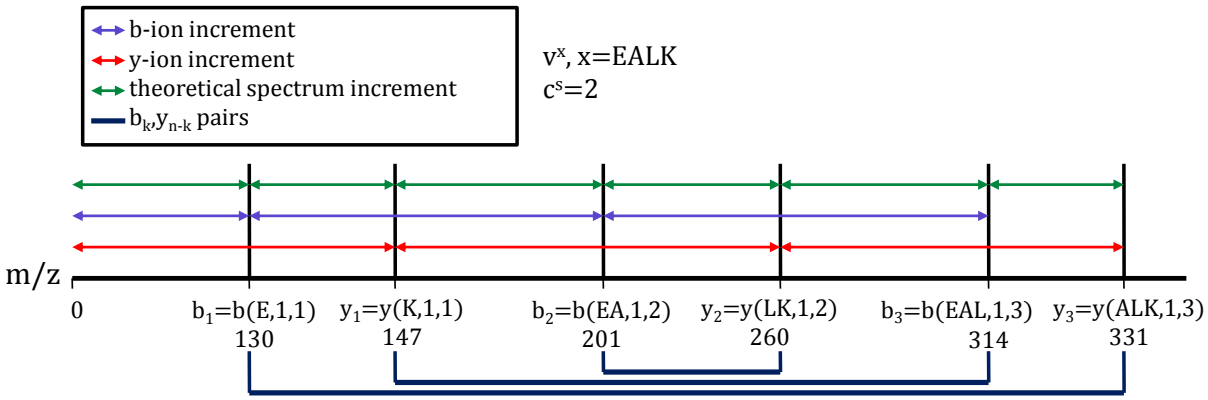


Figure 2.4: B- and y-ions of the peptide x = EALK assuming $c(x) = 2$ and integerized of the fragment ions. Thus, $c^b = c^y = 1$ for $k = 1, 2, 3$.

The second type of theoretical fragment ion corresponds to the loss of a molecular group off of a b- or y-ion. Three types of losses are commonly encountered: loss of carbon monoxide (CO), ammonia (NH_3), or water (H_2O). These molecules are lost from the side-chain (Figure 2.1(a)) of the fragment ion so that we may determine when such losses may occur based on the amino acids present in the fragment ion: CO may only be lost from a b-ion; NH_3 may only be lost if at least one aspartic acid (D), glutamic acid (E), serine (S), or threonine (T) is present; H_2O may only be lost if at least one glutamine (Q), lysine (K), arginine (R), or asparagine (N) is present. In practice, the intensity of the neutral loss peaks are typically much lower than the corresponding b- and y-ion peaks.

Some methods further add peaks adjacent (i.e., 1 Th away) to b-/y-ions and neutral losses, which are commonly referred to as *flanking peaks*. Such flanking peaks were originally used in the first database search method, SEQUEST [20], in order to account for imprecision of m/z measurements. In practice, however, it has been observed that flanking peaks are commonly detrimental to identification accuracy [65]. The various types of theoretical peaks are summarized in Table 2.1.

Theoretical peak	Formula
b-ion	$b(x,c,k) = \frac{\sum_{i=0}^{k-1} m(x_i)+c}{c}$
y-ion	$y(x,c,k) = \frac{\sum_{i=\tilde{n}-k}^{\tilde{n}} m(x_i)+18+c}{c}$
CO loss (a-ion)	$b(x,c,k) - 27.9949$
H_2O loss	$i - 18.0106$ for b-/y-ion i if residue contains $\{\mathrm{Q},\mathrm{K},\mathrm{R},\mathrm{N}\}$
NH_3 loss	$i - 17.0266$ for b-/y-ion i if residue contains $\{\mathrm{D},\mathrm{E},\mathrm{S},\mathrm{T}\}$
flanking peak	$i - 1$ Th for b-/y-ion or neutral loss i

Table 2.1: For peptide $x = x_0x_1 \ldots x_{n-1}$, $k < n-1$, and charge c, the different types of theoretical peaks commonly used in practice.

2.3 Tandem mass spectra resolution

Technological advances in the past decade have allowed significant improvements in precision when collecting tandem mass spectra. These precision improvements come in two forms, the first being significantly more accurate estimation of a spectrum's precursor mass and charge during the MS1 phase, and the second being significantly more accurate estimation of fragment ion locations in the MS2 phase. Datasets are typically characterized in terms of the resolution used during the data acquisition process: spectra collected using low-resolution in the MS1 phase and low-resolution in the MS2 phase are referred to as low-low (or LL) spectra, spectra collected using high-resolution in the MS1 phase and low-resolution in the MS2 phase are referred to as high-low (or HL) spectra, and spectra collected using high-resolution in the MS1 phase and high-resolution in the MS2 phase are referred to as high-high (or HH) spectra. While, in principle, it is possible to collect low-high spectra, this is uncommon in practice and such spectra are not further discussed.

2.3.1 Implications for peptide identification

These resolution improvements have two significant impacts on peptide identification via database search. The first such impact affects the candidate sets scored during a search, without directly affecting PSM scoring functions. When high-low or high-high data are collected, we may use a narrow precursor mass tolerance, w, when selecting peptide candidates to score. Compared to low-low spectra, where w is often a few thomsons and the corresponding number of candidates per spectra is typically on the order of hundreds to thousands, high-low and high-high data often use w narrow enough that candidates range in at most dozens per spectrum. This means that high-resolution MS1 data collection often equates to orders of magnitude faster search time compared to its low-resolution counterpart. Furthermore, beyond just faster search times, this also means that by scoring and ranking far fewer peptides, the likelihood that a random peptide scores higher than the true peptide is lower (assuming the true peptide is in the database). Thus, one would expect both faster and more accurate

peptide identification to accompany high-resolution MS1 acquisition.

The second impact directly affects PSM scoring functions, without affecting the candidate sets to be scored. When high-high data are collected, the measurement of fragment ion locations are incredibly accurate. Search algorithms may take advantage of the increased certainty in fragment ion locations by modeling their theoretical peaks with more certainty. For most search algorithms, this means finer-grained quantization of the theoretical and observed spectra. Where previously, for low-low and high-low data, the quantization bin-widths were ~ 1 Th, with high-high data the quantization bin-widths are often two orders of magnitude smaller, ranging from anywhere between 0.02 Th to 0.05 Th (though these numbers may change based on instrument and dataset). This directly impacts search accuracy, often yielding significantly more identifications at stringent error tolerances.

Chapter 3

PREVIOUS WORK

3.1 Scoring Functions

Many database search scoring functions have been proposed to identify peptides from an MS/MS experiment. As previously mentioned, practically all such scoring functions share the common thread of comparing a peptide's theoretical spectrum to an observed spectrum. In order to further improve a database search method's identification accuracy, several post-processing methods have been proposed which, accepting as input the PSMs of a database search method, recalibrate PSM scores. We now discuss in further detail both scoring functions and post-processors.

All mass spectrometry devices have a finite observable Th range. Let the maximum observable Th range of the mass spectrometer be $\bar{o} \in \mathbb{N}$. Many existing methods begin by quantizing or fixed bin-width thresholding and preprocessing an observed spectrum s. For the discussion that follows, let $z \in \mathcal{R}^{\bar{o}+1}$ be the vector resulting from preprocessing s and let $u \in \mathcal{R}^{\bar{o}+1}$ be the theoretical vector based on the theoretical spectrum of a given candidate peptide p (as described in Section 2.2).

3.1.1 SEQUEST's XCorr

The very first database search algorithm for peptide identification, SEQUEST [20], remains one of the most widely used tools in the field today. The elements of u correspond to hardcoded weights for the integerized fragment ions (i.e., indices of u); b- and y-ions are assigned weight 50, flanking peaks to b- and y-ions are assigned weight 25, and neutral losses are assigned weight 10. In order to compare u and z, SEQUEST utilizes the following scoring

function, called XCorr,

$$\text{XCorr}(s, x) = u^T z - \frac{1}{75} \sum_{\tau=-75}^{75} u^T z_\tau \tag{3.1}$$

$$= u^T (z - \frac{1}{75} \sum_{\tau=-75}^{75} z_\tau) = u^T z'. \tag{3.2}$$

XCorr thus consists of a foreground term (dot-product) minus a background (cross-correlation), where the background term is meant to penalize overfitting of the theoretical spectrum. By the linearity of XCorr, the calculation $z' = (z - \frac{1}{75} \sum_{\tau=-75}^{75} z_\tau)$ may be combined into the SEQUEST preprocessing of s so that, caching z' once per observed spectrum, scoring n candidate peptides requires $\mathcal{O}(n\bar{o})$ operations given dense theoretical spectrum representations and $\mathcal{O}(nm)$ operations given sparse theoretical spectrum representations, where m is the maximal number of nonzero theoretical peaks amongst the n candidates to be scored. As we'll see in subsequent sections, the study and analysis of XCorr remains an active area of research. XCorr is also closely related to log-likelihood of one of the graphical models heavily studied in this work, Didea (discussed in Sections 6.2 and 6.2.5).

3.1.2 X!Tandem

X!Tandem [14] employs a dot product between a binary theoretical spectrum and and a real-valued observed spectra. The resulting value is then multiplied by a factorial of the number of matched b-ions as well as a factorial of the number of the matched y-ions, which arise from assuming a hypergeometric distribution over fragment ion matches.

3.1.3 Parametrically calibrated scoring functions

Scoring functions often suffer from poor *calibration*, where PSM scores from different spectra are difficult to compare to one another. Thus, although a scoring algorithm may perform well ranking an observed spectrum's generating peptide above all other database peptides (the task of *relative ranking*), the scoring algorithm may perform poorly in terms of ranking target PSMs above decoy PSMs from other spectra (discussed in futher detail in Chapter 4).

As such, much recent work has focused on calibrating scoring functions by measuring the significance of a particular PSM score with respect to some distribution of scores.

Original attempts at developing calibrated scoring functions focused on assuming particular scoring functions followed specific parametric forms, beginning with the Open Mass Spectrometry Search Algorithm (OMSSA) [23]. OMSSA estimates p-values with respect to the count of the b- and y-ions of x present in z (i.e., a dot-product between z and a binary valued u) by fitting this count to a Poisson distribution with mean parameter derived from the properties of the observed spectrum. For SEQUEST, several parametric calibration methods have been proposed to calibrate XCorr scores. In [46], a Weibull distribution is fit to the distribution of XCorr scores per observed spectrum and p-values are calculated for top-scoring PSMs. More recent work, which we refer to as the XCorr E-value, has focused on fitting an exponential distribution to the candidate peptide XCorr scores per spectrum [21].

3.1.4 Nonparametrically calibrated scoring functions: MS-GF+ and XCorr p-values

The MasS Generating Function (MS-GF+) [43, 44] algorithm was the first database search algorithm to calculate exact, nonparametric p-values with respect to a base scoring function. In MS-GF+, the observed spectrum is heavily preprocessed and the theoretical vector u is Boolean valued. The base scoring function in MS-GF+ is then the dot-product $u^T z$. This dot-product is then calibrated by calculating its p-value with respect to the distribution of scores for all peptides with equal precursor m/z m^s,

$$\text{MS-GF+}(s, x) = \frac{1}{\left|\mathbb{P}^{m^s}\right|} \sum_{\alpha \in \mathbb{P}^{m^s}} \mathrm{p}_{\mathcal{D}}(\alpha) \mathbf{1}\{u_\alpha^T z \geq u^T z\}, \tag{3.3}$$

where $\mathbb{P}^{m^s}$ is the set of all possible peptides with equal precursor m/z ($\mathbb{P}^{m^s} = \{\alpha : m(\alpha) = m^s, \alpha \in \mathbb{P}\}$), u_α is the Boolean theoretical spectrum for $\alpha \in \mathbb{P}^{m^s}$, $\mathrm{p}_{\mathcal{D}}(\cdot)$ is a distribution over amino acids derived from the amino acid frequencies of the database $\mathcal{D}$, and amino acids are assumed i.i.d. such that $\mathrm{p}_{\mathcal{D}}(x) = \prod_{i=0}^{n-1} p_{\mathcal{D}}(x_i)$. For arbitrary scoring functions, the exact p-value calculation in Equation 3.3 is computationally expensive (i.e., without any structure to exploit, we would have to score 20^n peptides, assuming a maximum peptide length of

n). However, the linearity of the underlying base scoring function allows Equation 3.3 to be computed efficiently using dynamic programming.

Exact, nonparametric XCorr p-values

Exact, nonparametric p-values have also been computed for XCorr [33]. Like MS-GF+, the exact distribution of peptide scores given an observed spectrum is calculated. The linearity of XCorr, as seen in Equation 3.1, is exploited to compute p-values (as in Equation 3.3) via dynamic programming. This approach makes no assumptions on the per spectrum distribution of scores, and thus is less susceptible to inaccurate modeling of score distributions compared to the parametric methods discussed in Section 3.1.3, at the cost of more computation; parametric methods assume a closed form so that the overall complexity does not change from scoring candidate peptides per observed spectrum using some scoring function, whereas additional computation is necessary to calculate the dynamic programming matrix necessary for exact nonparametric p-values. The exact details of the dynamic programming are now discussed.

Special care must be taken so that the dynamic programming matrix may be exactly computed. Specifically, the dynamic program must be written exclusively in terms of either the sequence of b-ions or y-ions. We arbitrarily assume that the dynamic programming recurrence is defined in terms of the b-ions. Ingeniously, as detailed in [33], given a precursor mass m and assuming a precursor charge of 2+, for an integerized b-ion i, the sibling integerized y-ion occurs at $m - (i + 1) - 18 = m - i - 19$. The contribution of b-ion i's sibling y-ion is then added to $z(i)$, adding another layer of preprocessing (the additional preprocessing compute is neglible to total complexity). Note that, due to the assumption of a precursor mass and the corresponding collapse of the sibling y-ion into the preprocessed observed spectrum, this calibration must be run for each integerized precursor mass within the precursor mass tolerance range. Furthermore, the contribution of b-ion i's neutral losses (carbon monoxide, ammonia, and water) are added to i, as well as all other higher charged related b-ions and y-ions when the precursor charge is greater than 2+. At the end of this process, we have

Algorithm 1 Compute the exact distribution of XCorr scores via dynamic programming.

Assume a given observed spectrum, s.
Let a^+ and a^- be upper bound and lower bounds, respectively, on $\text{XCorr}(s, x)\ \forall\ x \in \mathbb{P}$.
Assume a scoring function granularity of δ and precursor mass m.
Define $r = \lceil \frac{a^+ - a^-}{\delta} \rceil$.
Initialize the dynamic programming matrix $A \in \mathcal{R}^{m^+} \times \mathcal{R}^r$ to all zeros.
Denote the set of amino acids as $\mathcal{A}$.
for $i = 1, \ldots, m$ **do**
    for $j = 1, \ldots, r$ **do**
        for $\alpha \in \mathcal{A}$ **do**
            $A(i, j) = A(i - m(\alpha), j - \text{round}(\hat{z}(i - m)/\delta)) + 1;$
        end for
    end for
end for
Initialize $Z = 0$. // Normalization constant
for $j = 1, \ldots, r$ **do** // Calculate normalization constant
    $Z+ = A(m, j);$
end for
for $j = 1, \ldots, r$ **do**
    $A(m, j) = Z;$
end for
Return $A(m, :);$

a reprocessed observed vector $\hat{z}$ and binary theoretical $\hat{u}$ with nonzero entries at charge 1+ b-ion indices, so that $\text{XCorr}(s, x) = u^T z' = \hat{u}^T \hat{z}$. Defining $b(x, 1, 0) = 1$, from Equation 2.3 we have the recurrence

$$b(x, 1, k) = \sum_{i=0}^{k-1} m(x_i) + 1 = m(x_i) + b(x, 1, k-1). \tag{3.4}$$

We may then calculate the dynamic programming matrix and distribution of XCorr scores (the last column in the dynamic programming matrix) for all $x \in \mathbb{P}$ using Algorithm 1. The overall complexity of this process is $\mathcal{O}(mr|\mathcal{A}|)$, where m is the precursor mass, r is the number of bins for which scores are quantized, and $\mathcal{A}$ is the set of amino acids.

3.2 Recalibrating PSMs via post-processors

Far different from scoring functions (Section 3), post-processing methods accept as inputs PSMs from a scoring function run over a dataset in order to recalibrate their scores for improved identification accuracy. The first such post-processor, PeptideProphet [15, 8], recalibrated scores via a linear discriminant function. Utilizing unlabeled target and decoy PSM scores, PeptideProphet first computed linear discriminant scores for PSMs then recalibrates these scores via a probability computed using the expectation-maximization algorithm [15, 8]. The calibration method was unsupervised. In MS/MS, supervision refers to utilizing labels denoting whether a PSM is correct or incorrect. We may accurately assign negative labels to decoy PSMs as we know no decoy could have generated an observed spectrum. Target labels come with no such guarantess as we cannot, with certainty, say that a target PSM generated an observed spectrum. Thus, in the case of semi-supervised training, target PSMs are denoted as the positive class and decoy PSMs are denoted as the negative class.

Semi-supervised post-processing methods were subsequently shown to achieve superior performance to the original PeptideProphet. The first such semi-supervised approach was Percolator [38]. Given the output target and decoy PSMs of a scoring algorithm and several features derived for each PSM (such features typically include charge, peptide mass, score, as well as many others), Percolator utilizes a support vector machine (SVM) to learn a

classifier between target PSMs and decoy PSMs. Critical to performance is identifying a subset of target and with a stringent q-value threshold (typically 1%) and utilizing this set of high-confidence target PSMs as well as all decoy PSMs to learn the support vectors. After learning a classifier and identifying a high-confidence set of PSMs, this overall process is repeated until the set of high-confidence PSMs does not change. Upon convergence, PSMs are rescored based on their distance from the learned hyperplane. Subsequent work further improved on the idea utilized in Percolator, utilizing neural networks with a specific loss function (ramp loss) for improved target/decoy classification [66, 67]. Despite the increase in accuracy afforded by these more recent works, Percolator remains one of the most popular post-processors amongst the MS/MS community.

Recent work has focused on the manner in which the subset of high-confidence target and decoy PSMs are used during training in Percolator [25]. To avoid overfitting, which is highly probable if we do not separate the train and test sets, a nested cross-validation procedure is introduced, wherein the set of all PSMs is split into three disjoint sets. Each split is further split into three disjoint sets, where two of the sets are used to identify high-confidence PSMs (i.e., use decoy labels to assign q-values) and the remaining set is used to test. This process is iterated to convergence, and the resulting split PSMs are normalized and merged.

With Percolator being one of the most popular post-processors actively used, much work has focused on its utilization with other popular scoring algorithm. In the original paper, Percolator was described for use with XCorr and relevant features. Subsequent work showed how Percolator and appropriate features may be used with Mascot for improved Mascot accuracy [12]. Similar work was done to combine X!Tandem and Percolator [75]. Most recently, MS-GF+ PSMs and a carefully curated set of features were combined with Percolator for improved identification accuracy [26].

We later show how graphical models may be used to extract features and further improve Percolator post-processing performance for XCorr p-values and MS-GF+ (Section 9.1).

Chapter 4

ASSESSING PEPTIDE IDENTIFICATION ACCURACY

A significant challenge in evaluating the quality of a spectrum identification algorithm is the absence of a "ground truth" data set where the generating spectrum is known for each observed spectrum. We therefore follow the standard approach of using *decoy peptides* (which in our case correspond to randomly shuffled versions of each real *target* peptide) to estimate the number of incorrect identifications in a given set of identified spectra. In this work, targets and decoys are scored separately and used to estimate the number of identified matches at a particular *false discovery rate* (FDR), i.e., the fraction of spectra improperly identified at a given significance threshold.

We estimate FDR using the target-only variant of target-decoy competition (T-TDC), wherein the top-scoring target and decoy PSMs compete with one another and only the higher scoring of the two is retained per spectrum. For a particular score threshold, t, FDR is then calculated as the proportion of decoys scoring better than t to the number of targets scoring better than t. In [40], it was proven that T-TDC is asymptotically accurate, as compared to another commonly used MS/MS estimate of FDR [39], which was shown to underestimate the true FDR. Because the FDR is not monotonic with respect to the underlying score, we instead use the *q-value*, defined to be the minimum FDR threshold at which a given score is deemed to be significant. Because datasets containing more than 10% incorrect identifications are generally not practically useful, we only plot q-values in the range $[0, 0.1]$.

4.1 Benchmark datasets

The yeast (*Saccharomyces cerevisiae*) and worm (*C. elegans*) data sets were collected using tryptic digestion followed by acquisition using low-resolution precursor scans and low-resolution

Table 4.1: Summary of presented datasets.

Data set	Spectra	Charges	w
Worm-1	22,693	1–3	±3.0 Th
Worm-2	21,862	1–3	±3.0 Th
Worm-3	20,011	1–3	±3.0 Th
Worm-4	23,697	1–3	±3.0 Th
Yeast-1	35,236	1–3	±3.0 Th
Yeast-2	37,641	1–3	±3.0 Th
Yeast-3	35,414	1–3	±3.0 Th
Yeast-4	35,467	1–3	±3.0 Th
Malaria	12,594	2–6	±50 ppm

fragment ions. Each dataset exhibited charge 1+, 2+, and 3+ spectra. Each search was performed using a ±3.0 Th tolerance for selecting candidate peptides. Peptides were derived from proteins using tryptic cleavage rules without proline suppression (i.e., cleave at arginine or lysine irregardless of the presence of proline at the cleavage site) and allowing no missed cleavages. A single fixed carbamidomethyl modification was included. Further details about these data sets, along with the corresponding protein databases, may be found in [38].

The malaria (*Plasmodium falciparum*) sample was digested using Lys-C, labeled with an isobaric tandem mass tag (TMT) relabeling agent, and collected using high-resolution precursor scans and high-resolution fragment ions. The data set consists of 12,594 spectra with charges ranging from 2+ through 6+. Searches were run using a 50 ppm tolerance for selecting candidate peptides, a 0.03 Th fragment mass tolerance, a fixed carbamidomethyl modification, a fixed TMT labeling modification of lysine and N-terminal amino acids. Further details may be found in [74].

The relevant features of all datasets are summarized in Table 4.1.

4.2 Benchmark algorithm settings

All XCorrs and XCorr p-values were collected using Crux v2.1.16567 [52]. XCorrs and XCorr p-values were collected using `tide-search` with flanking peaks not allowed, resulting in the best performance for each respective method. X!Tandem version 2013.09.01.1 was used, with PSMs scored by E-value. MS-GF+ scores were collected using MS-GF+ version 9980, with PSMs ranked by E-value. Default parameters were used (unless otherwise stated) except that, to make a fair comparison with other methods, isotope peak errors were not allowed and, to ensure the set of scored candidate peptides does not differ from other methods, methionine clipping is turned off.

When searching the Malaria dataset, the following settings were used to take advantage of the high-resolution fragment ions. XCorr utilized a fragment ion error tolerance of 0.03 (`mz-bin-width=0.03`) and initial fragment ion offset of 0.5 (`mz-bin-offset=0.5`). XCorr p-values are currently not designed to take advantage of the increased fragment ion resolution and so were collected with default settings. MS-GF+ was run with `-inst 1`, denoting a high resolution fragment ion instrument. X!Tandem was run with fragment monoisotopic mass error equal to 0.03 Daltons.

4.3 Benchmark performance

For a given method, we refer to measure of performance consisting of the number of identified spectra at varying q-values as absolute ranking. The absolute ranking curves of MS-GF+, XCorr p-values, XCorr, and X!Tandem (utilizing the search settings in Section 4.2) are plotted in Figures 4.1 and 4.2 for all datasets described in Section 4.1.

All benchmarks are commonly used searched algorithms with free implementations available (XCorr, XCorr p-value, X!Tandem are even available as open source). The majority of these search algorithms offer state-of-the-art performance and a valuable benchmark to compare the accuracy of the new methods employing graphical models and described herein. Details regarding the exact settings for each benchmark algorithm are available in Section 4.2

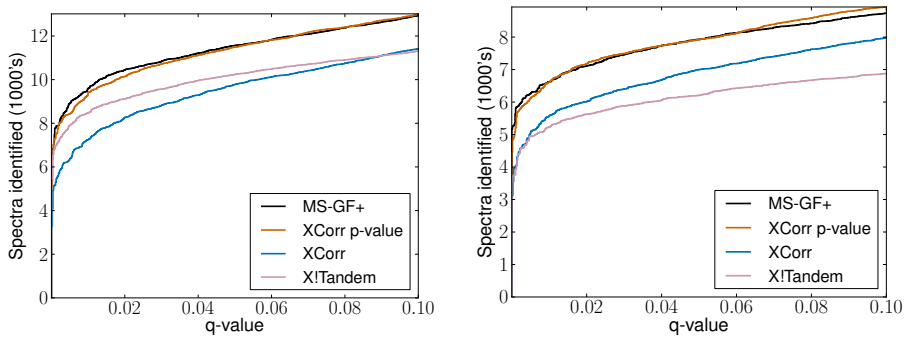
and details regarding the presented datasets are available in Section 4.1.

From Figure 4.1, it is clear that the p-value methods MS-GF+ and XCorr p-value significantly outperform the other benchmarks. This is a common thread in MS/MS scoring functions; due to the nature of the target-decoy strategy for computing FDR, well calibrated scoring functions often perform significantly better than uncalibrated scoring functions. This is best exemplified in the difference in identified spectra for all q-values between uncalibrated XCorr (in blue) and its calibrated counterpart XCorr p-value (in orange). Such performance gains are the reason that much research in recent years has focused on calibrated MS/MS scoring functions [43, 44, 33, 41]. However, the assumption necessary to calculate exact, nonparametric p-values in MS-GF+ and XCorr p-value (seen in Equation 3.4) does not carry over to high-resolution fragment ions, where typical quantization bin-widths are orders of magnitude smaller than 1 Th, so that low-resolution fragment ions are assumed for XCorr p-value and MS-GF+ for high-high data. In this regime, it is the case that these calibrated methods may be outperformed by methods which capitalize on the high-resolution fragment ions (Figure 4.2).

4.4 Practical considerations when benchmarking search algorithms

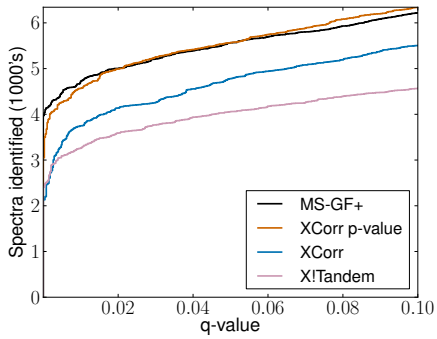
Critical to a fair comparison amongst different methods is that they score exactly the same peptide candidates per spectrum. Special care must be taken when comparing MS/MS search algorithms to one another as the (often unexpected) behavior of several search algorithms makes an exact comparison (i.e., exactly same set of candidate peptides and observed spectrum charges) difficult. While the following settings may not reflect the best settings for the many available MS/MS scoring algorithms, they are necessary to ensure that performance is not biased by information utilized in one scoring algorithm and not the others being compared.

Digestion rules are deterministic and thus should yield exactly the same behavior and thus candidate peptides. However, some search algorithms implement non-standard digestion rules. For instance, for a tryptic digest, MS-GF+ produces peptides both with and without proline suppression. In contrast, all other search algorithms support trypsin digestion either

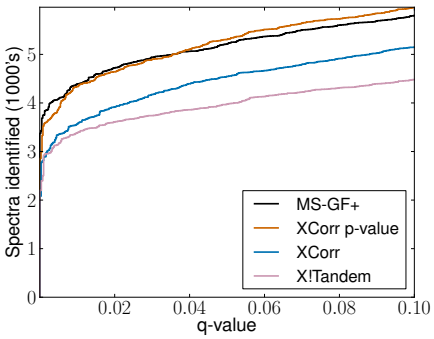


(a) Worm-1

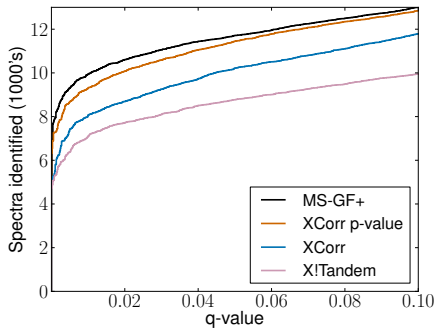
(b) Worm-2



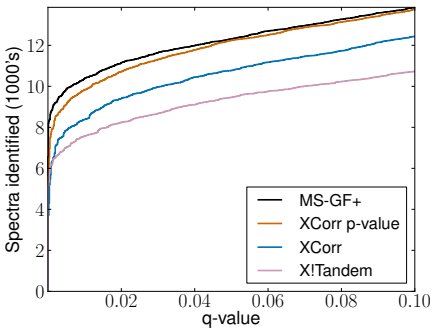
(c) Worm-3



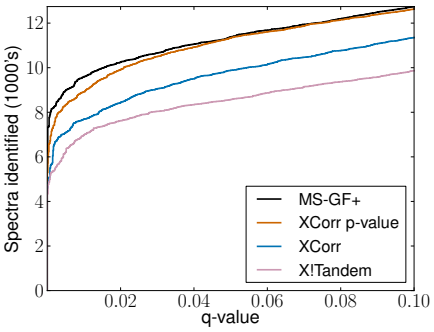
(d) Worm-4



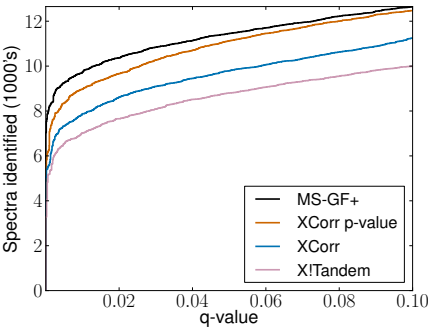
(e) Yeast-1



(f) Yeast-2



(g) Yeast-3



(h) Yeast-4

Figure 4.1: MS-GF+, XCorr p-value, XCorr, and X!Tandem performance for the eight low-low datasets.

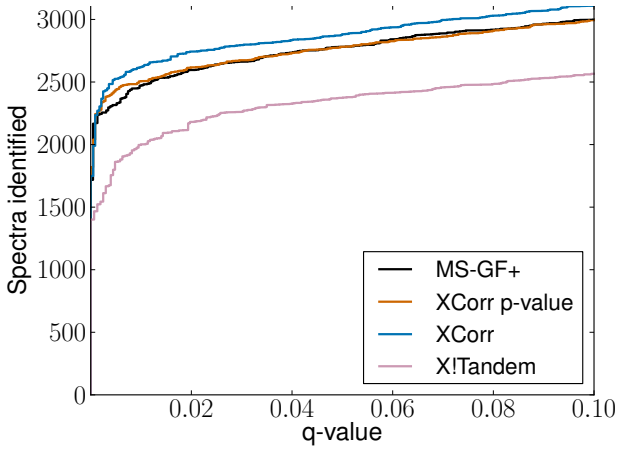


Figure 4.2: MS-GF+, XCorr p-value, XCorr, and X!Tandem for the high-high Malaria data.

with proline suppression or without, but not a mix of the two. In order to ensure the same peptides are scored for all competitors, it is best to pre-digest the protein database and construct a FASTA file of the digested peptides. This also means that, for a tryptic digest, proline suppression must be ignored to ensure MS-GF+ adheres to the expected digestion rules.

In practice, proteins with a leading methionine often see this amino acid clipped. As such, all search algorithms should adhere to either clipping or not clipping a lead methionine. Note that when methionine is clipped, both the clipped and unclipped peptide are typically added to the peptide database. When using a pre-digested peptide database, methionine should be clipped to guard against peptides which would not be considered in the original digested database.

Observed spectra are assigned charge states, with some being assigned multiple charges due to charge uncertainty during data acquisition. Some search algorithms, such as MS-GF+, ignore observed charge information and instead assume, per spectrum, all allowable charges specified for the search. In contrast, most search engines only search a spectrum with their specified observed charge states. Modeling unobeserved charge states may lead to degraded search accuracy, as the search engine is considering incorrect peptide candidates, increasing

the chance of a random PSM outscoring valid peptides. To guard against this behavior, a dataset may be split into several different files, each of which contains all spectra of a unique charge (i.e., any spectrum with an observed 1+ charge). Each individual file may then be run by specifying only its unique charge be searched. The separate charge searches may then be merged to obtain the unbiased result.

Some search algorithms do not allow the specification of missed cleavages (such as MS-GF+), they search possible missed cleavages and internally choose the best value suited for their scoring function. This, of course, results in significantly different peptide candidate sets than other search algorithms when searched with a specific number of missed cleavages. To ensure that the candidate peptide sets are the same across all different search algorithms, pre-digesting the protein database with either no missed cleavages or an infinite number of missed cleavages is necessary.

Although all these procedures significantly complicate a typical search, they are absolutely critical when benchmarking a new scoring function to ensure a fair comparison. Once again, we note that these parameters are not reflective of the best settings for all scoring algorithms, but are necessary to ensure an unbiased comparison amongst methods.

Chapter 5

GRAPHICAL MODELS

5.1 Overview of Bayesian networks

Consider that you wish to model some phenomena such as the weather, the rise and fall of stocks, or a speaker's voice in order to perform *inference*, the task of answering queries of interest in regards to the phenomena. However, due to either measurement error, incomplete understanding of the phenomena, or a host of other factors, there is uncertainty with respect to the phenomena's behavior. We often build probabilistic models to account for such uncertainty and perform probabilistic inference. As we'll see, naive probabilistic inference is often intractable, but *probabilistic graphical models* offer the potential to perform such inference efficiently.

We motivate the need for efficient probablistic inference algorithms with the following simple example. Assume that you have n discrete random variables $X_1, X_2, \ldots, X_n$, with joint distribution $p(X_1, X_2, \ldots, X_n)$. For simplicitly, assume that all random variables have the same cardinality, $r = |X_1| = |X_2| = \cdots = |X_n|$. Now, consider the problem of calculating the marginal distribution of a single random variable, say X_n, which may be naively achieved as follows

$$p(X_n) = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{n-1}} p(X_1, X_2 = x_2, X_3 = x_3, \ldots, X_n = x_n). \tag{5.1}$$

To simplify notation, when it is unambigious we describe the probability of an event $p(X_i = x_i)$ as $p(x_i)$ and a set of random variables as $X_{i:j} = \{X_i, X_{i+1}, \ldots, X_{j-1}, X_j\}$, where $i \leq j$. Now, even in the case that $r = 2$ and all the random variables of interest are Bernoulli, Equation 5.1 would require $\mathcal{O}(2^n)$ operations, a computation which is intractable for large n. However, if there are conditional independences amongst the random variables, the computation in

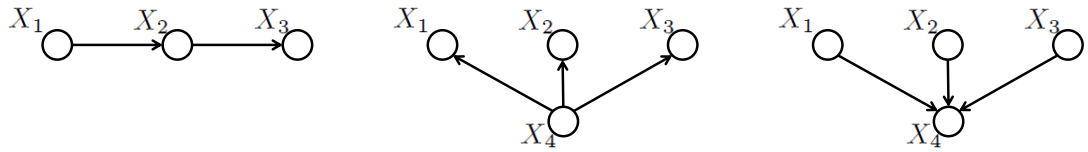
Equation 5.1 may be greatly simplified.

For instance, letting $X_0 = \emptyset$, and for $i = 1, \ldots, n$, if we knew that $X_{0:i-1}$ and $X_{i+1:n}$ were conditionally independent given X_i (denoted as $X_{0:i-1} \perp\!\!\!\perp X_{i+1:n} | X_i$), then using basic rules of probability, Equation 5.1 simplifies as

$$
\begin{aligned}
p(X_n) &= \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{n-1}} p(x_1, x_2, x_3, \ldots, X_n) \\
&= \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{n-1}} p(x_1) p(x_2|x_1) p(x_3|x_2, x_1) \ldots p(X_n|x_{n-1}, \ldots, x_1) \quad \text{by the chain rule} \\
&= \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{n-1}} p(x_1) p(x_2|x_1) \ldots p(X_n|x_{n-1}) \quad \text{since } X_{0:i-1} \perp\!\!\!\perp X_{i+1:n} | X_i \qquad (5.2) \\
&= \sum_{n-1} p(X_n|x_{n-1}) \sum_{n-2} p(X_{n-1}|x_{n-2}) \cdots \sum_{x_3} p(x_4|x_3) \sum_{x_2} p(x_3|x_2) \sum_{x_1} p(x_1) p(x_2|x_1). \qquad (5.3)
\end{aligned}
$$

Such a set of random variables, where $X_{0:i-1} \perp\!\!\!\perp X_{i+1:n} | X_i$, is known as a *Markov chain*. We obtain 5.3 by rearranging 5.2 such that sums and the quantities which reference the summands are pushed as far to the right as possible. Equation 5.3 requires $\mathcal{O}(nr^2)$ operations, and we've gone from a computation which grows exponentially in the number of random variables to one which grows quadratically in r, all by exploiting conditional independence amongst the random variables. Such exploitation of conditional independences for efficient inference is formalized by probabilistic graphical models (PGMs), wherein a graph represents the conditional independence properties (called *Markov properties*) amongst a set of random variables. The graphical nature of PGMs affords one to visually build and intuitively reason about a model, while providing a formal set of rules to determine the conditional influence between random variables and perform efficient inference. Our discussion will focus on *Bayesian networks*, wherein Markov properties are defined over *directed acyclic graphs* (DAGs). For a description of *Markov Random Fields*, wherein Markov properties are defined over undirected graphs, the reader is directed to [5, 49, 48].

A Bayesian network (BN) consists of a DAG $G = (V, E)$, the vertices V of which correspond to a set of random variables and the directed edges E of which correspond to possible



(a) Length 3 Markov chain. (b) Naive Bayes model. (c) BN containing a v-structure.

Figure 5.1: Examples of Bayesian networks.

conditional dependencies in the graph. Directed edges are said to point from parent to child. A BN encodes how the joint distribution of a set of random variables *factorizes*, i.e. decomposes into a product of conditional distributions. The manner in which this factorization occurs may be intuitively read from the BN, such that each *factor* involved in the overall product is the probability of a random variable conditioned on its parents. For example, the joint distribution of the length 3 Markov chain in Figure 5.1(a) factorizes as $p(X_1, X_2, X_3) = p(X_1)p(X_2|X_1)p(X_3|X_2)$, the joint distribution of the Naive Bayes model in Figure 5.1(b) factorizes as $p(X_1, X_2, X_3, X_4) = p(X_1|X_4)p(X_2|X_4)p(X_3|X_4)p(X_4)$, and the joint distribution of the BN in Figure 5.1(c) factorizes as $p(X_1, X_2, X_3, X_4) = p(X_1)p(X_2)p(X_3)p(X_4|X_1, X_2, X_3)$.

5.1.1 Conditional independence in Bayesian networks

One powerful feature of a BN is the ability to easily determine the influence random variables have on one another. Conditional independence amongst a set of random variables may be determined by inspecting the BN, the formal semantics of which are referred to as *d-separation* and by which we determine the independence between variables conditioned on other variables in the network. For example, for the Markov chain in Figure 5.1(a), all paths from X_1 to X_3 go through X_2. If we condition on X_2, i.e. if we know X_2 with certainty, then X_1 and X_3 are conditionally independent, as no information could flow from X_1 to X_3. In this case, X_2 acts as a *separator* for X_1 and X_3 and, conditioned on X_2, X_1 and X_3 are independent and

said to be d-separated. For the Naive Bayes model in Figure 5.1(b), X_4 is a separator for variables X_1, X_2, X_3.

Figure 5.1(c) depicts a very important case in d-separation; notice that X_1 and X_2 both have edges to a common child, X_4, but do not have edges to one another, visually forming a v. This is known as a *v-structure*, and in this case if we condition on X_4, X_1 and X_2 are conditionally dependent; if we do not condition on X_4, X_1 and X_2 are independent. Note the contrast between this and the network in Figure 5.1(b) which contains no v-structures; for the BN in Figure 5.1(b), conditioning on X_4 results in X_1, X_2, X_3 all being conditionally independent, and otherwise X_1, X_2, X_3 are conditionally dependent. Thus, two variables are d-separated in a BN if for each directed path from one variable to the other, the common child in all v-structures are not conditioned on and at least one separator along the path is conditioned on.

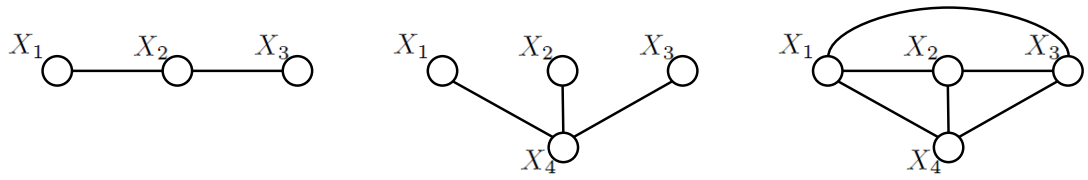
A set of random variables whose conditional independencies respect the Markov properties of a BN are said to be an *I-map* (independence map), and a set of variables for which no further Markov properties hold other than those of the BN is said to be a *perfect map*. As an example, a set of independent random variables X_1, X_2, X_3 are certainly an I-map for the chain in Figure 5.1(a) since, for such random variables, it is true that $X_1 \perp\!\!\!\perp X_3 | X_2$, but these variables do not form a perfect map as $X_1 \perp\!\!\!\perp X_3 | \emptyset$ is not a Markov property of the BN.

5.1.2 Inference in Bayesian networks

Marginalization by the elimination algorithm

As we saw in the length n Markov chain described in Equation 5.3, the factorization of the joint distribution combined with rearranging and distributing the summations allowed us to calculate $p(X_n)$ in $\mathcal{O}(nr^2)$ time. The algorithm underlying this set of operations is known as *variable elimination*, the *elimination algorithm*, or simply *eliminate*, and may be used to efficiently marginalize any number of variables in a PGM. Marginalization of a random variable may be thought as eliminating this node from the graph, and the order in which we

eliminate variables is known as an *elimination ordering*, where the random variable whose marginal distribution we wish to obtain is not a part of this order(or placed last in the order and not actually eliminated). It turns out that the algorithmic complexity in a BN is highly dependent on the elimination ordering.



(a) Moralized length 3 Markov chain.

(b) Moralized Naive Bayes model.

(c) BN containing a v-structure, moralized.

Figure 5.2: Moralized Bayesian networks from Figure 5.1.

We begin the elimination algorithm by *moralizing* the graph $G = (V, E)$, such that the parents of all child nodes share edges amongst each other and, subsequently, the directionality of all edges is dropped. Thus, moralization forms *cliques* between child variables and all their parents, where a clique is a fully connected subset of nodes $V' \subseteq V$. After moralization, we obtain an undirected graph $\mathcal{G} = (V, \mathcal{E})$, where the set of vertices V remain unchanged from G. The moralized graphs for the BNs in Figures 5.1(a), 5.1(b), 5.1(c) are displayed, respectively, in Figures 5.2(a), 5.2(b), 5.2(c).

Next, for each variable in a given elimination ordering, we collect all factors which reference that variable, then marginalize over this variable. The distribution resulting from this marginalization is referred to as a *message* and is a function of a subset of the remaining variables to be eliminated, the number of which determine the complexity of the marginalization. Take as an example the length 5 Markov chain depicted in Figure 5.3(a) and moralized in Figure 5.3(b), and assume the elimination ordering (X_3, X_4, X_2, X_1) for which we are calculating $p(X_5)$. The resulting redistributed sums over the factorized joint

distribution are then

$$\begin{aligned} p(X_5) &= \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_3} p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_3)p(X_5|x_4) \\ &= \sum_{x_1} p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_4} p(X_5|x_4)\sum_{x_3} p(x_3|x_2)p(x_4|x_3). \end{aligned} \tag{5.4}$$

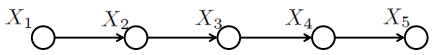
Once calculated after each marginalization, each message is passed along to the next variable to be eliminated and possibly joined to the product of factors for the next marginalization. For arbitrary variables $a, b, c, d, e, \dots$, we denote these messages as $\mu_{a\to b}(c, d, e, \dots)$, where a is the variable most recently eliminated, b is the next variable to be eliminated which recieves the incoming message, and $c, d, e, \dots$ are the variables for which this message is a function of and have yet to be eliminated. Armed with this notation, we pick up where we left off in Equation 5.4,

$$\begin{aligned} p(X_5) &= \sum_{x_1} p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_4} p(X_5|x_4)\sum_{x_3} p(x_3|x_2)p(x_4|x_3) \\ &= \sum_{x_1} p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_4} p(X_5|x_4)\mu_{x_3\to x_4}(x_2, x_4) \\ &= \sum_{x_1} p(x_1)\sum_{x_2} p(x_2|x_1)\mu_{x_4\to x_2}(x_2, X_5) \\ &= \sum_{x_1} p(x_1)\mu_{x_2\to x_1}(x_1, X_5) \\ &= \mu_{x_1\to x_5}(X_5) \end{aligned}$$

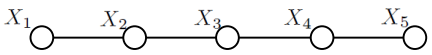
Note that the summations to compute messages $\mu_{x_3\to x_4}(x_2, x_4), \mu_{x_4\to x_2}(x_2, X_5), \mu_{x_2\to x_1}(x_1, x_5)$ are over 3-dimensional tables and will each require $\mathcal{O}(r^3)$ operations. Already, we can see that we went from quadratic complexity in r in Equation 5.3 now to cubic complexity in r. Each step of elimination behaves as a manipulation of the induced graph over the yet-to-be eliminated variables, such that variables which are not connected prior to eliminating a variable but are involved in this variable's marginalization afterwards share edges; indeed, we view these variables as being "coupled" together as the resulting outgoing message is

a function of all of them. Such extra edges are called *fill-in edges* and, after every step of eliminate, we obtain a new graph over the remaining variables along with any additional fill-in edges over these variables.

As an example, after computing $\mu_{x_3 \to x_4}(x_2, x_4)$, node x_3 and all edges to and from this node are removed. In the original graph $\mathcal{G}$, X_2 and X_4 are not connected but are after eliminating X_3, and thus we have the resulting graph in Figure 5.3(c) where the fill-in edges are colored in red. The graphs resulting from each step of the elimination ordering (X_3, X_4, X_2, X_1) are displayed in Figures 5.3(c), 5.3(d), 5.3(e), 5.3(f).



(a) Length 5 Markov chain.



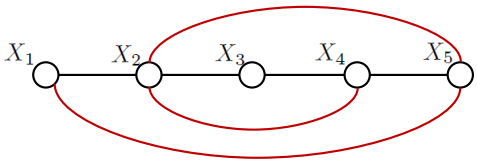
(b) Moralized Markov chain.

(c) Eliminate X_3.

(d) Eliminate X_4.

(e) Eliminate X_2.

(f) Eliminate X_1.



(g) Reconstituted graph $\mathcal{G}$, with fill-in edges colored in red.

Figure 5.3: Eliminate run on a length 5 Markov chain with elimination order (X_3, X_4, X_2, X_1). The input Bayesian network, Figure 5.3(a), is first moralized, the output graph of which is displayed in Figure 5.3(b). Figures 5.3(c), 5.3(d), 5.3(e), 5.3(f) show the resulting graphs after each variable elimination, where fill-in edges are colored red. The reconstituted graph corresponding to this elimination order is displayed in Figure 5.3(g).

Denoting the set of fill-in edges as $\mathcal{E}'$, the reconstituted graph $\mathcal{G}' = (V, \mathcal{E} \cup \mathcal{E}')$ is known as the *triangulated graph*, and the manipulation of $\mathcal{G}$ to create $\mathcal{G}'$ (i.e. running eliminate) is known as *triangulation*. Notice that in the triangulated graph in Figure 5.3(g), the fill in edges added by eliminating variables X_3, X_4, and X_2 result in the cliques $\{X_2, X_3, X_4\}$, $\{X_2, X_4, X_5\}$, and $\{X_1, X_2, X_5\}$, respectively, where each of these cliques are of size 3. It is no coincidence that the summations to compute messages $\mu_{x_3 \to x_4}(x_2, x_4), \mu_{x_4 \to x_2}(x_2, X_5), \mu_{x_2 \to x_1}(x_1, X_5)$ require $\mathcal{O}(r^3)$ operations. Indeed, the maximum clique size in the triangulated graph dictates the overall complexity of inference, and determining an elimination ordering which minimizes the triangulated graph's maximum clique size is necessary to optimize inference complexity.

Unfortunately, determining such an elimination ordering for general graphs turns out to be NP-hard and heuristics are most often used [48, 34, 37]. While this is certainly disappointing news, it turns out that if the moralized graph, $\mathcal{G}$, is a tree, an optimal elimination ordering (there may be many) corresponds to successively eliminating leaf variables. Such a strategy results in no fill-in edges and, due to $\mathcal{G}$ being a tree, the maximum clique size being 2 and overall inference complexity $\mathcal{O}(|V|r^2)$. Such is the reason inference in the length n Markov chain found in Equation 5.2 required $\mathcal{O}(nr^2)$ operations, as the elimination order corresponding to this regrouping of factors and redistribution of summations is $(X_1, X_2, X_3, \ldots, X_{n-1})$.

From eliminate to the sum-product algorithm

We've seen that eliminate may be run with an optimal elimination ordering if the moralized graph is a tree. Correspondingly, we restrict our attention to trees in the current discussion, coming back to the general case in Section 5.1.2. We've seen how eliminate may be used to compute any marginal distribution in a BN. However, what if we were interested in all marginal distributions in the BN? We could certainly run eliminate $|V|$ times, once for each variable in the network. Such a strategy has complexity $\mathcal{O}(|V|^2 r^2)$ and proves to be far from optimal; we can, in fact, calculate all marginals in $\mathcal{O}(|V|r^2)$ time, the same as eliminate! The algorithm for which we achieve this linear complexity in $|V|$ is known as the *sum-product algorithm*. We will be trading time for memory, as using eliminate to calculate a

single marginal requires $\mathcal{O}(|V|r^2)$ time and $\mathcal{O}(r^2)$ memory, whereas using the sum-product algorithm to calculate all marginals requires $\mathcal{O}(|V|r^2)$ time and $\mathcal{O}(|V|r^2)$ memory.

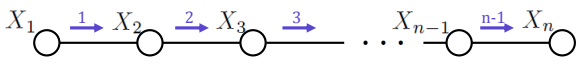
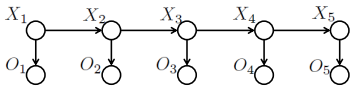


Figure 5.4: Message passing in a Markov chain to compute $p(X_n)$.

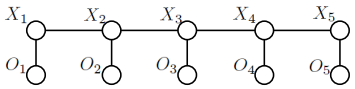
We can view eliminate as a message passing procedure where, given a tree, messages flow from the leaves towards the last variable to be eliminated. Such a message passing scheme used to compute $p(X_n)$ in a moralized length n Markov chain is depicted in Figure 5.4, where the index above a blue edge denotes the order in which a message is passed. Thus, message $\mu_{x_1 \to x_2}(x_2)$ is first passed, then $\mu_{x_2 \to x_3}(x_3)$, and so on. In such a chain, it turns out that the messages are recursively defined as $\mu_{x_i \to x_{i+1}}(x_{i+1}) = \sum_{x_i} \mu_{x_{i-1} \to x_i}(x_i) p(x_{i+1}|x_i)$. The base case of this recursion is $\mu_{x_0 \to x_1}(x_1) = p(x_1)$, so that we have $\mu_{x_i \to x_{i+1}}(x_{i+1}) = \sum_{x_i} p(x_i) p(x_{i+1}|x_i) = p(x_{i+1})$. Thus, after a single forward pass consisting of all messages flowing up from the leaves, we can compute all marginals in a chain. However, for general trees, this is not the case.

The sum-product algorithm begins by assigning an arbitrary node as the root in the moralized tree $\mathcal{G}$. To compute all marginals in $\mathcal{G}$, we will need both a forward pass of messages propagating from the leaves up to the root, as well as a backward-pass of messages flowing from the root back down to the leaves. Computing a backward message is the same as computing a forward message; to compute $\mu_{x_i \to x_j}(x_j)$, all factors involving x_i are multiplied together, along with all incoming messages save for $\mu_{x_j \to x_i}(x_i)$, and subsequently node x_i is marginalized. An alternate message formulation involves initializing all messages to unity, multiplying all incoming messages and a division by $\mu_{x_j \to x_i}(x_i)$, though these two forms are equivalent [48] and may be used accordingly based on implementation constraints.

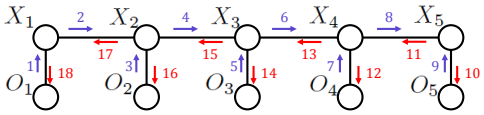
However, for the marginals to be correct after these two passes, each node must adhere to the *message passing protocol* (MPP), which states that a node only send messages once it has



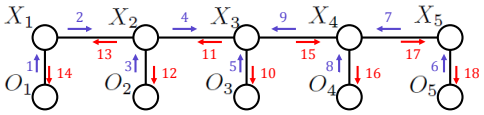
(a) BN whose moralized graph is a tree.



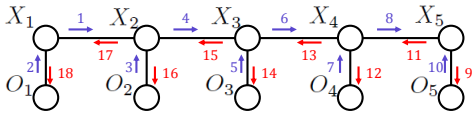
(b) Moralized graph of the BN.



(c) Root node is X_5, MPP is respected.



(d) Root node is X_3, MPP is respected.



(e) Root node is X_5, MPP is violated.

Figure 5.5: Flow of messages in a BN whose moralized graph is a tree. The designated root for messages in Figures 5.5(c) and 5.5(e) is X_5, and the designated root for messages in Figure 5.5(d) is X_3. Thus, all forward messages, depicted by blue arrows, flow toward to the root and all backward messages, depicted by red arrows, propagate from the root.

received all of its incoming messages. Note that the MPP encapsulates the aforementioned optimal flow of both forward and backward messages in a tree as, by the MPP, the leaves are the only nodes which may send messages at the start of the algorithm and backward messages may only be sent once all forward messages reach the root. It can be proven that if the MPP is respected, then the marginals obtained once the forward and backward passes have completed converge to the true marginals of the variables in the BN [48]. Figures 5.5(c) and 5.5(d) depict passed messages which respect the MPP. Figure 5.5(e) shows passed messages for which the MPP is violated, as $\mu_{x_5 \to o_5}(o_5)$ (message 9 in red) is passed before $\mu_{o_5 \to x_5}(x_5)$ (message 10 in blue) and $\mu_{x_1 \to x_2}(x_2)$ is passed before $\mu_{o_1 \to x_1}(x_1)$. The sum-product algorithm requires computing twice the number of messages as variable elimination, and thus runs in $\mathcal{O}(|V|r^2)$ time. Furthermore, all these messages must be retained so that the sum-product algorithm requires $\mathcal{O}(|V|r^2)$ memory, whereas in variable elimination, we need only retain the most recently computed message and thus eliminate requires only $\mathcal{O}(r^2)$ memory.

The junction tree algorithm for general graphs

When $\mathcal{G}$ is not a tree, then the *junction tree algorithm* may be used. In this case, the graph is first triangulated using the eliminatation procedure. A junction tree is then formed, where nodes correspond to cliques in the triangulated graph and edges are shared between two nodes if there is a non-empty interesection between the nodes' corresponding cliques. The sum-product algorithm is then run over the resulting junction tree and used to compute the distributions over the cliques. For further details, the reader is directed to [5, 48].

The Viterbi algorithm

Variable elimination, the sum-product algorithm, and the junction-tree algorithm all fall under the umbrella of algorithms which utilize message passing for inference in PGMs, and these algorithms are collectively referred to as *belief propagation* (BP). The algebraic structures which allowed us to derive these message passing algorithms were that sums and products are both associative and commutative, and products are distributive over sums. The operations

of sums and products together form a *commutative semi-ring* [22], and thusly the moniker for the sum-product algorithm. It turns out that these message passing algorithms may be used for any commutative semi-ring. One such semi-ring that is often encountered in practice is max-product, which occurs when we wish to determine the configuration of random variables which maximizes the joint probability in a BN, and the message passing algorithm used to determine this configuration is known as the *max-product* or *Viterbi* algorithm. The maximizing assignment of random variables computed via the Viterbi algorithm is referred to as the *Viterbi path* or *most probable explanation* in the BN.

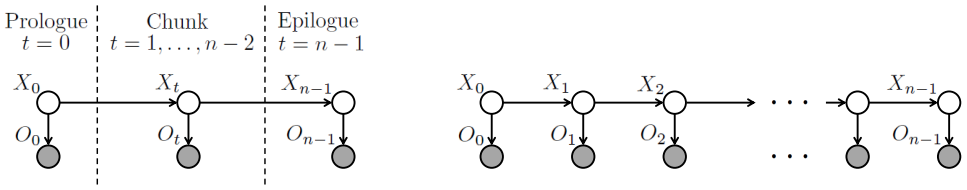
5.1.3 Dynamic Bayesian networks

Armed with our knowledge of BNs, we now focus our discussion on *dynamic Bayesian networks* (DBNs), BNs for which the number of variables is allowed to vary. In keeping with the notation in [5, 6], we begin all sequences with index 0. DBNs are most often used to model sequential data, such as speech [6], biological [32], or economic [29] sequences. Each time instance of the sequence is modeled by a *frame*, which consists of a set of random variables and edges amongst these variables. A DBN is often defined in terms of a *template* where the first frame is called the *prologue*, the final frame is called the *epilogue*, and the *chunk* consists of multiple frames in between the prologue and epilogue. As an example, the template for a *hidden Markov model* (HMM) is depicted in Figure 5.6(a), where t denotes the frame index. The sequence of random variables $X_0, X_1, \ldots, X_{n-1}$ are *hidden*, i.e. stochastic, and are typically referred to as the *hidden layer*. The sequence of random variables $O_0, O_1, \ldots, O_{n-1}$ are *observed*, i.e., each variable takes on a single value, and are typically referred to as the *observed layer* or as *evidence*.

The observed layer is used to model a sequence of observations, most often measurements or features derived from measurements of the signal of interest. The hidden layer is typically used to model uncertainty with respect to the observed layer or the random process which we do not have direct access to but from which we assume the observations to be produced. These two ideas, the hidden layer and evidence, often underly DBNs and the algorithms

developed for them [5, 55].

When a length n sequence is to be modeled, the chunk is then *unrolled*, i.e., dynamically expanded, to fill frames $t = 1, \ldots, n-2$, shown for our HMM in Figure 5.6(b). Thus, in order to perform inference, we may simply expand the template, for which we produce a single "wide" BN. After unrolling, we may then perform BP inference over the resulting BN. However, the sequences typically modeled can be quite long. For instance, the number of frames in speech signals is typically on the order of hundreds or thousands. As such, highly optimized BP algorithms may be run specifically for DBNs, such as the *forward* and *backward recursions* [5, 55] (also known as the α and β recursions, respectively), which allow us to perform inference without first unrolling the graph but rather recursively computing forward and backward messages, thus avoiding the memory overhead of the instantiated unrolled graph. For extremely long sequences for which even the forward and backward recursions do not provide enough memory savings to perform inference, as is the case in genomic data [32] where the number of frames is typically on the order of millions, algorithms exist which allow inference time to be traded for memory; these algorithms are the *island* [9] and *archipelagos* [1] algorithms.



(a) Template for an HMM.

(b) HMM unrolled.

Figure 5.6: Template for a hidden Markov model and the template unrolled.

As a concrete modeling example, assume we know nothing about the day-to-day state of the weather being either sunny, rainy, or snowy, but we observe whether a particular coworker brought an umbrella to work each day and that the weather does not change drastically

day-to-day. We could model this using an HMM and infer the most likely state of the weather for each day. Each frame would correspond to a day, our observations would be binary and denote whether our coworker brought in an umbrella, and each X_i would be a multinomial denoting one of 3 possible weather states. The conditional dependence between the X_i and X_{i+1} in the model encodes our knowledge that the weather cannot drastically change between adjacent days. Once obtaining a sequence of observations, we could then use the Viterbi algorithm to find the most likely sequence of weather states. Furthermore, given a sequence of observations, we would also be able to learn all conditional probabilities in the model, $p(X_{i+1}|X_i)$ and $p(O_i|X_i)$, in both unsupervised (i.e., without labelled observations) and supervised settings via the expectation-maximization algorithm [15, 8].

5.2 Virtual Evidence

Virtual evidence [58, 2] is a versatile mechanism by which, given a BN, we may model the conditional distributions of observed leaf nodes using nonnegative distributions which need not be normalized for a great deal of probabilistic quantities commonly of interest. Formally, for a BN $G = (V, E)$ with edge set E and vertex set V, let $O \subseteq V = \{o_1, \ldots, o_n\}$ be a set of observed leaf nodes, and $H = V \setminus O$. For any $S \subseteq V$, denote the set of parents for all variables in S as π_S. The conditional distribution of $o_i \in O$ is then $\mathrm{p}(o_i|\pi_{o_i}) = \frac{\psi_i(o_i,\pi_{o_i})}{Z_i}$, where $\psi_i : \mathcal{R}^{|\pi_{o_i}|+1} \to \mathcal{R}_+$, Z_i is a normalizing constant, and $\mathcal{R}_+$ is the set of nonnegative reals. The joint distribution over G is thus

$$\begin{aligned}
\mathrm{p}(O, H) =& \mathrm{p}(O, \pi_O)\mathrm{p}(H|O, \pi_O) \\
=& \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \mathrm{p}(o_i|\pi_{o_i}) \\
=& \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \frac{\psi_i(o_i, \pi_{o_i})}{Z_i} \\
=& \frac{1}{Z}\mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i}),
\end{aligned} \tag{5.5}$$

where $Z = \prod_{i=1}^{n} Z_i$. Note that $\mathrm{p}(H|O, \pi_O)$ may further factorize depending on the BN.

We now show that, for many practical applications of interest, we do not need the normalization constants $Z_1, \ldots, Z_n$ when computing the Viterbi path, the probability of evidence, or posteriors of G. We define the following quantity, which will prove useful throughout our discussion,

$$\mathrm{p}'(O, H) = Z\mathrm{p}(O, H) = \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i}).$$

Posterior probabilities

Consider the case where we are interested in posterior probabilities in G, i.e., for $H' \subseteq H$, we would like to compute $\mathrm{p}(H'|O)$. We thus have

$$\begin{aligned}
\mathrm{p}(H'|O) =& \frac{\mathrm{p}(H', O)}{\mathrm{p}(O)} \\
=& \frac{\sum_{X \in H \setminus H'} \frac{1}{Z} \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i})}{\sum_{X \in H} \frac{1}{Z} \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i})} \\
=& \frac{\sum_{X \in H \setminus H'} \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i})}{\sum_{X \in H} \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i})} \\
=& \frac{\sum_{X \in H \setminus H'} \mathrm{p}'(O, H)}{\sum_{X \in H} \mathrm{p}'(O, H)}.
\end{aligned}$$

Thus, the normalization constants are not necessary to compute posteriors over the hidden variables.

Probability of evidence and Viterbi score

The *probability of evidence*, or *score*, is the quantity computed after summing over all hidden variables in G. It is often used to score different sets of observations. For instance, consider that we have m sets of observations $O^i = \{o_1^i, \ldots, o_n^i\}$ for $i = 1, \ldots, m$, where each of the sets may contain observed information regarding a peptide, observed spectrum, or both. Now, consider we'd like to score and rank each O^i, using G, as $\mathrm{p}(O^i)$ in order to choose the maximum scoring set of observations $O^* = \mathrm{argmax}_{O^i, i=1,\ldots,m} \mathrm{p}(O^i)$. This is exactly the scenario we have when performing an MS/MS database search for the model in Section 6.1.

From Equation 5.5, we thus have

$$\begin{aligned} O^* &= \underset{O^i, i=1,\ldots,m}{\text{argmax}}\ \mathrm{p}(O^i) \\ &= \underset{O^i, i=1,\ldots,m}{\text{argmax}} \sum_{X \in H} \frac{1}{Z} \mathrm{p}(H|O^i, \pi_O) \prod_{j=1}^{n} \psi_j(o_j, \pi_{o_j}) \\ &= \underset{O^i, i=1,\ldots,m}{\text{argmax}}\ \frac{1}{Z} \sum_{X \in H} \mathrm{p}(H|O^i, \pi_{O^i}) \prod_{j=1}^{n} \psi_j(o_j, \pi_{o_j}) \\ &= \underset{O^i, i=1,\ldots,m}{\text{argmax}} \sum_{X \in H} \mathrm{p}'(O, H). \end{aligned}$$

This is the case since $\mathrm{p}(O^i, H) \propto \mathrm{p}'(O^i, H)$ and, since Z is constant with respect to the sum, $\sum_{X \in H} \mathrm{p}'(O, H) \propto \sum_{X \in H} \mathrm{p}(O, H)$. Thus, we need not worry about the normalizing constants $Z_1, \ldots, Z_n$ if we need only make decisions based on the relative scores between sets of observations.

Now, instead of the probability of evidence, consider we would like to score each O^i with their Viterbi score. We thus have

$$\begin{aligned} O^* &= \underset{O^i, i=1,\ldots,m}{\text{argmax}}\ \max_{X \in H} \frac{1}{Z} \mathrm{p}(H|O^i, \pi_{O^i}) \prod_{j=1}^{n} \psi_j(o_j, \pi_{o_j}) \\ &= \underset{O^i, i=1,\ldots,m}{\text{argmax}}\ \frac{1}{Z} \max_{X \in H} \mathrm{p}(H|O^i, \pi_{O^i}) \prod_{i=j}^{n} \psi_j(o_j, \pi_{o_j}) \\ &= \underset{O^i, i=1,\ldots,m}{\text{argmax}}\ \max_{X \in H} \mathrm{p}'(O^i, H) \end{aligned}$$

so that, once again, our ranking of each O^i based on Viterbi score does not depend on the normalzing constants $Z_1, \ldots, Z_n$.

Viterbi path

Consider we'd like to compute the Viterbi path H^* of G given O. We thus have

$$\begin{aligned} H^* &= \operatorname*{argmax}_{X \in H} \frac{1}{Z} \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i}) \\ &= \operatorname*{argmax}_{X \in H} \mathrm{p}(H|O, \pi_O) \prod_{i=1}^{n} \psi_i(o_i, \pi_{o_i}) \\ &= \operatorname*{argmax}_{X \in H} \mathrm{p}'(O, H). \end{aligned}$$

Thus, computing the Viterbi path does not require the normalzing constants $Z_1, \ldots, Z_n$.

Discussion

Virtual evidence allows a rich set of distributions to fit under the umbrella of graphical models. We use virtual evidence in almost every DBN discussed herein, touching upon each one of the probabilistic quantities discussed above. For Didea (Section 6.2), we utilize virtual evidence to weight observed peaks, computing a posterior with which to score peptides. In Section 6.1, we utilize a DBN employing virtual evidence to model any linear MS/MS scoring function to score and rank peptides. For DRIP (Section 6.3), wherein we compute both Viterbi paths and Viterbi scores, the conditional Gausian distribution over the observations is a form of virtual evidence.

Chapter 6

GRAPHICAL MODELS FOR PEPTIDE IDENTIFICATION

Armed with a firm understanding of GMs, in this section we show how many of the scoring functions described in Section 3.1 may be represented utilizing GMs. One incredible advantage of GMs is that, given a general purpose graphical models inference engine, such as the Graphical Models ToolKit (GMTK) [3], we may rapidly develop and prototype widely different models, without the need to rewrite and debug large quantities of code from scratch. GMTK is a highly optimized software package which offers a rich set of variable modeling options (including virtual evidence, conditional Gaussian/Gamma leaf nodes, and switching parents [4]), supports both max-product and sum-product inference, and provides a wide variety of approximate inference algorithms. Representing existing scoring algorithms as GMs, we leverage approximate inference options in GMTK to significantly improve the runtime of linear scoring functions (Section 7). We furthermore describe, as they were previously introduced in the literature, two GMs, Didea and DRIP, designed for highly accurate MS/MS spectrum identification.

In Didea, the time series being modeled is the sequence of a peptide's amino acids (i.e., an amino acid is observed in each frame). The entire preprocessed observed spectrum is observed in each frame. In successive frames, the sequence of b- and y-ions are computed and used as indices into the preprocessed observed spectrum via Virtual Evidence (Section 5.2). A single random variable, corresponding to the amount to shift the observed spectrum by, is hidden in the first frame. Finally the posterior of the shift random variable being zero is used to score each peptide candidate. This posterior is similar in form to XCorr (Equation 3.1), consisting of a foreground score, which only involves the non-shifted observed spectrum, minus a background score, which involves an average over shifted versions of the observed

spectrum.

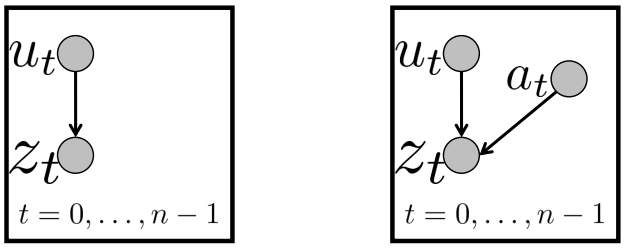
In DRIP, the time series being modeled is the sequence of observed spectrum peaks (i.e., each frame in DRIP corresponds to an observed peak). The entire theoretical spectrum is hidden and traversed. Insertions (spurious peaks) and deletions (absent theoretical peaks) are explicitly modeled via latent random variables in each frame. The Viterbi path of the sequence of insertions and deletions is calculated and the corresponding Viterbi score (the log-likelihood of the most probable configuration of hidden variables) is used to score a peptide candidate. Thus, DRIP may be thought of as aligning the theoretical and observed spectra in a manner similar to string alignment and edit distance.

Didea and DRIP differ significantly from one another (in fact, they are inverted in the sequences that they model; sequence of a peptide's amino acids versus the sequence of observed peaks) with various strengths and weaknesses relative to one another. For instance, performing Viterbi decoding in DRIP to score peptides returns a wealth of information regarding each PSM, which we show may be used to derive features for improved Percolator post-processing. Viterbi decoding is not an effective means to score peptides in Didea (the only non-deterministic random variable in the model is the shift variable in the prologue). Furthermore, while both models afford efficient parameter estimation (in stark constrast to many existing scoring functions which do not allow or support effective parameter estimation with most relying on hardcoded hyperparameters), learning parameters in DRIP lead to much larger performance gains than in Didea. However, Didea has much more determinism than DRIP and thus runs much faster in practice.

We begin our discussion with a relatively simple class of GMs, the log-likelihoods of which correspond to MS/MS scoring functions linear in the theoretical and observed spectra. We then make our way to the more involved Didea, also discussing its scoring function's explicit relationship to SEQUEST's XCorr. We conclude with DRIP, arguably the most complicated of any GMs discussed herein. However, unlike all other GMs and MS/MS scoring methods discussed, DRIP does not quantize or fixed-width threshold the observed spectrum, instead posing the alignment of theoretical and observed spectra as an inference problem. We note

that all models (including the sophisticated DRIP model) were developed and tested using GMTK for the diverse tasks of speeding up database search time and improving search accuracy, illustrating the modeling power and generality afforded by casting problems as inference in a GM. For the various models tested, the reader is provided specific GMTK command lines to exhibit specific GM inference algorithms.

6.1 Linear scoring functions



(a) Mixture model representing linear scoring functions

(b) Mixture model representing linear scoring functions with theoretical peak weights

Figure 6.1: Mixture models for linear scoring functions, utilizing plate notation where the boxed random variables are repeated n frames. Shaded nodes denote observed variables.

Many MS/MS scoring functions are linear (XCorr, the base score for MS-GF+, X!Tandem). Utilizing Virtual Evidence, we may represent any MS/MS linear scoring function as a mixture model. Let $u_0, u_1, \ldots, u_{n-1}$ be a sequence of theoretical peaks and s be a preprocessed observed spectrum (a length m vector whose ith element is $s(i-1)$). If we construct an equal m length vector u which is non-zero at elements $u_0, u_1, \ldots, u_{n-1}$, then many scoring functions are small variants of $u^T s$. Let u_t be an observed random variables such that $p(u_t) = 1$ and z_t be a virtual evidence child such that $p(z_t = 1|u_i) \propto e^{s(u_i)}$. The log-likelihood of the mixture

model in Figure 6.1(a) is then equivalent to $u^T s$, since

$$\begin{aligned}\mathrm{p}(u_0, \ldots, u_{n-1}, z_0, \ldots, z_{n-1}) &= \prod_{t=0}^{n-1} \mathrm{p}(u_t)\mathrm{p}(z_t|u_t) = \prod_{t=0}^{n-1} \mathrm{p}(z_t|u_t) \\ &\propto \exp(\sum_{t=0}^{n-1} s(u_i)) = e^{u^T s},\end{aligned}$$

where the use of Virtual Evidence allows us to ignore computing the constant of proportionality (discussed in Section 5.2).

However, many of the MS/MS linear scoring functions weight the different types of theoretical peaks with varying coefficients. For instance, in XCorr, b- and y-ions are assigned weight 50, neutral losses are assigned weight 10, and flanking peaks are assigned weight 25. For our sequence of theoretical peaks $u_0, u_1, \ldots, u_{n-1}$, we may think of each peak as having an associated weight $w_0, w_1, \ldots, w_{n-1}$ so that, constructing a new non-boolean vector $\hat{u}$, we have $\hat{u}^T s = \sum_{i=0}^{m-1} w_i s(u_i)$. As such, we may incorporate these weights by ion type utilizing the mixture model in Figure 6.1(b), where $p(z_t|u_t, a_t) \propto e^{w_t s(u_t)}$ and the likelihood is

$$\begin{aligned}\mathrm{p}(u_0, u_1 \ldots, a_0, a_1 \ldots, z_0, z_1, \ldots) &= \prod_{t=0}^{n-1} \mathrm{p}(u_t)\mathrm{p}(a_t)\mathrm{p}(z_t|u_t, a_t) = \prod_{t=0}^{n-1} \mathrm{p}(z_t|u_t, a_t) \\ &\propto \exp(\sum_{t=0}^{n-1} w_t s(u_t)) = \exp(\hat{u}^T s)).\end{aligned}$$

As before, utilizing Virtual Evidence, we ignore the constant of proportionality during inference, so that the log-likelihood of the mixture model is exactly the quantity that we are interested in.

This relatively simple model illustrates the great deal of modeling power and flexibility afforded using GMs. With a simple change in the observed spectrum preprocessing and theoretical peak weights, using the GM in Figure 6.1(b) allows us to model any one of the scoring functions for SEQUEST [20], X!Tandem [14], Morpheus [73], and the base scores of MS-GF+ [44] and OMSSA [24]. Utilizing such a general representation, any work regarding this GM generalizes, in principle, to any scoring function in the same class. For instance, in Chapter 7, we significantly speed up XCorr utilizing the GM in Figure 6.1(b). Using the

same techniques, we would expect similar speed-ups for other linear MS/MS scoring functions. This also has a profound impact for parameter estimation. A majority of methods do not afford learning parameters, but rather rely on hardcoded parameter values. Establishing general training methods for the GM in Figure 6.1(b) means we may learn parameters for the wide range of possible scoring functions. The effectiveness of this training will depend on the model used, but any theoretical guarantess regarding the learned parameters will hold.

6.1.1 Exact, nonparametric p-values of linear scoring functions

In order to further illustrate the incredible modeling power afforded by GMs, we present a DBN with which we can compute exact, nonparametric p-values with respect to the XCorr scoring function and may be easily adapted for any linear scoring function. As in Algorithm 1, assume a scoring function granularity of δ and precursor mass m. For a peptide x, we overload m so that $m(x)$ is the peptide's mass. For a given observed spectrum s, assume we've processed s to produce $\hat{z}$, as detailed in Section 3.1.4, so that for peptide x with boolean vector of charge 1+ b-ions b, XCorr is $\text{XCorr}(s, x) = b^T \hat{z}$.

The graph of the DBN is in Figure 6.2. Let $X_i \in \mathcal{A}$ be a uniformly distributed, random amino acid for frames $i = 0, \ldots, n-1$. $B_0, \ldots, B_n$ are random variables corresponding to all possible b-ions for all peptides with precursor mass m, such that $B_0 = 1$ and $B_n = m + 1$. For $i = 1, \ldots, n-1$, $\text{p}(B_i = b_{i-1} + m(x_{i-1}) | B_{i-1} = b_{i-1}, X_{i-1} = x_{i-1}) = 1$. Z_i is a random variable such that $\text{p}(Z_i = \text{round}(\hat{z}(b_i)/\delta) | B_i = b_i, \hat{z}) = 1$, i.e., it measures the current b-ion's contribution to the overall XCorr score. Given $\hat{z}$, the conditional distribution $\text{p}(Z_i | B_i, \hat{z})$ may be computed in $\mathcal{O}(|B_i|)$ time. Q_i is a random variable which accumulates all score contributions across all frames, such that $Q_0 = 0$ and $\text{p}(Q_i = Q_{i-1} + Z_i | Q_{i-1}, Z_i) = 1$ for $i = 1, \ldots, n$.

Using this model, we efficiently consider all peptides with precursor mass equal to m. For each peptide in the set, their sequence of b-ions is calculated. Each b-ion's contribution to the their peptide's XCorr score is accessed via Z_i and added to the total XCorr score up to that frame, Q_i. As a whole, we compute the set of possible XCorr scores for peptides with

precursor mass m. Denoting the set of all vertices in the graph as V, the distribution over all possible XCorr scores given the observed spectrum and precursor mass is then the posterior

$$p(Q_n|s,m) = p(Q_n|\hat{z},m) = \sum_{V \setminus Q_n} \mathrm{p}(V|\hat{z},m),$$

which may be efficiently computed using the junction tree algorithm. We may then easily compute p-values $\sum_q \mathrm{p}(Q_n = q|\hat{z},m)\mathbf{1}\{q > \mathrm{XCorr(s,x)}\}$ for a peptide x such that $m(x) = m$.

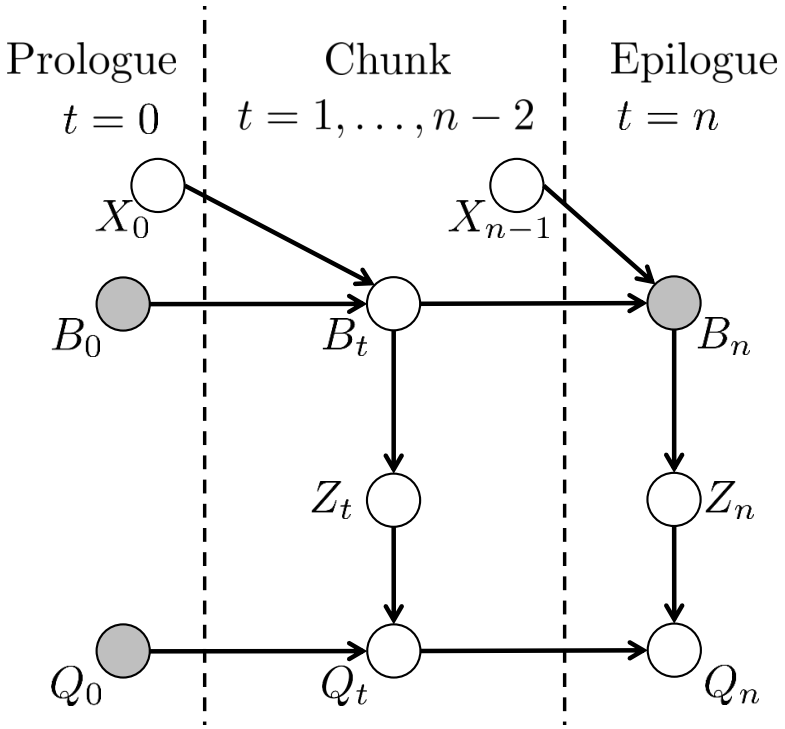


Figure 6.2: DBN template for exact, nonparametric p-values of linear scoring functions.

Although we used the DBN in Figure 6.2 to compute exact, nonparametric p-values for XCorr, p-values for any linear scoring function may be computed by simply changing the preprocessing of the observed spectrum. Thus, we may use this DBN to compute p-values for the other linear scoring functions mentioned in this work, X!Tandem [14], Morpheus [73], the base score of OMSSA, and the base score of MS-GF+ (which thus correspond to the p-values computed in MS-GF+, barring any unpublished tweaks in the closed source

implementation). Utilizing this model opens such MS/MS linear scoring functions to the wide breadth of algorithms and techniques accompanying graphical models. For instance, the Viterbi algorithm may be used to determine the most probable peptide sequence given the observed spectrum and precursor mass.

6.2 Didea

Didea [63] is a DBN based scoring function, each frame of which corresponds to one amino acid from the candidate peptide sequence. The observed spectrum is preprocessed, quantized, and observed in each frame. Let $z \in \mathcal{R}^{\bar{o}}$ be the output of Didea's preprocessing of s. Didea's scoring function is similar in form to XCorr. The b- and y-ions are calculated in successive frames and used as indices into shifted versions of z. Random access to the preprocessed observed spectrum is granted utilizing virtual evidence. A single hidden variable in the first frame, corresponding to the number of units to shift z, is then marginalized in order to compute a conditional log-likehood probability consisting of a foreground score and background score(much like Equation 3.1), where the foreground score is the log-likelihood of the model and the background score is the log of the denominator of the aforementioned posterior. A single hyperparameter (λ in the discussion to follow), controlling the reweighting of peak intensities was previously learned via grid search. We later detail how this parameter may be efficiently learned in Section 8.2.

6.2.1 Log-likelihood

The graph of Didea is displayed in Figure 6.3. Let $\tau_t \in [-L, L]$ be an integer random variable in frame $t = 0, \ldots, |x| - 1$, such that τ_0 is uniform and $\mathrm{p}(\tau_t = \tau_0) = 1$ for $0 < t \leq \tilde{n}$, where $\tilde{n} = |x| - 1$. As with SEQUEST, denote z_{τ_t} as the vector resulting from shifting z by τ_t units. Let $C^s \in \{2, 3\}$ be a uniform random variable modeling the possible precursor charges (i.e., possible observed spectrum charges), and let C be a uniform random variable over the set of possible b- charges for C^s. Thus, when $C^s = 2$, we have $C \in \{1\}$, and when $C^s = 3$, we have $C \in \{1, 2\}$. The random variable C'_t denotes a y-ion's charge and, as discussed

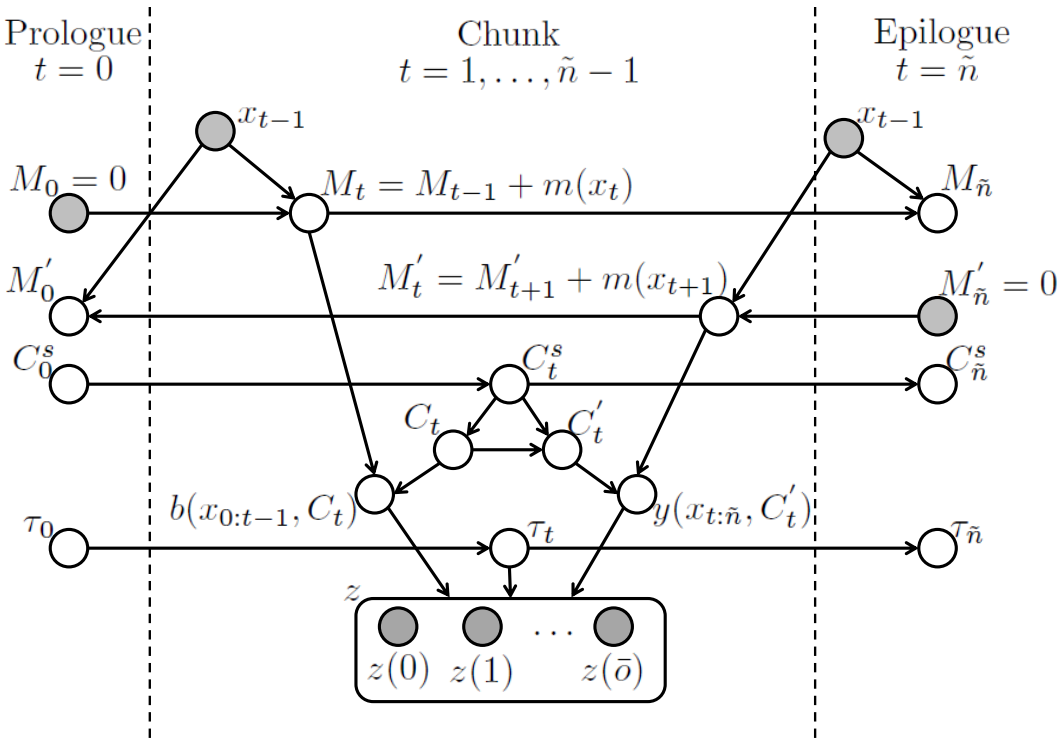


Figure 6.3: Graph of Didea. Black edges denote deterministic functions and blue edges denote switching parents. M_t and M_t' are the prefix and suffix masses, respectively, calculated recursively in successive frames and used to calculate b- and y-ions in the chunk frames. The entire preprocessed observed spectrum, z, is copied in each frame of the chunk. Note that, by the defined recursion, $M_{n^x} = m(x) = M_0'$.

in Section 2.2, is a deterministic function of the precursor and b-ion charges, such that $p(C_t' = (c^s - c)|C_t^s = c^s, C_t = c) = 1$.

Shaded variables are observed, i.e., take their values in each frame with probability 1, and unshaded variables are hidden (random). The amino acids of a peptide are observed in each frame after the prologue. The variable M_t successively accumulates the prefix masses of the peptide, while the variable M_t' successively accumulates the suffix masses of the peptide. M_t and M_t' are thusly used, along with the charge variables, to calculate the b- and y-ions of the peptide in the chunk. The variable τ_0 is hidden in the prologue and deterministically copied by its descendents in successive frames, such that $p(\tau_t = \tau|\tau_0 = \tau) = 1$ for $t > 0$. With τ_t and the b- and y-ions as parents to z, shifted versions of the preprocessed spectrum intensities are accessed.

For observed spectrum shift $\tau \in [-L, L]$, the log-likelihood of Didea is then

$$\begin{aligned}
\log \mathrm{p}(\tau_0 = \tau, x, z, C^s = c^s) &= \log \mathrm{p}(\tau_0 = \tau)\mathrm{p}(c^s)\mathrm{p}(x, z|\tau_0 = \tau, c^s) \\
&= \log \mathrm{p}(\tau_0 = \tau) + \log \mathrm{p}(c^s) + \log \mathrm{p}(x, z|\tau_0 = \tau, c^s) \\
&= -\log|\tau_0| - \log|C^s| + \log \prod_{t=1}^{\tilde{n}} \Big(\prod_{c=1}^{c^s-1} \mathrm{p}(c) z_\tau(b(x_{0:t-1}, c)) z_\tau(y_{t:\tilde{n}}, c^s - c) \Big) \\
&= -\log|\tau_0| - \log|C^s| + \log \prod_{t=1}^{\tilde{n}} \Big(\prod_{c=1}^{c^s-1} \mathrm{p}(c) z_\tau(b(x_{0:t-1}, c)) z_\tau(y_{t:\tilde{n}}, c) \Big) \\
&= -\log|\tau_0| - \log|C^s| + \\
&\quad \sum_{t=1}^{\tilde{n}} \Big(\sum_{c=1}^{c^s-1} (\log \mathrm{p}(c) + \log z_\tau(b(x_{0:t-1}, c) + \log z_\tau(y(x_{t:\tilde{n}}, c)) \Big) \quad (6.1)
\end{aligned}$$

Note that when $C^s = 2$, $\mathrm{p}(C = 1) = 1$ and Equation 6.1 simplifies to

$$\log \mathrm{p}(\tau_0 = \tau, x, z, C^s = 2) = -\log|\tau_0| - \log|C^s| + \sum_{t=1}^{\tilde{n}} \Big(\log z_\tau(b(x_{0:t-1}, 1) + \log z_\tau(y(x_{t:\tilde{n}}, 1)) \Big), \quad (6.2)$$

where the sum involves the logarithms of the binned, reweighted observed peaks accessed by the b- and y-ions of x. Thus, the behavior of Didea changes depending on the precursor charge C^s, in that when $C^s = 2$, C is deterministic, but when $C^s = 3$, C is a random variable.

6.2.2 Scoring function under assumed precursor charge

Assuming precursor charge c^s, Didea computes a conditional log-likelihood

$$\begin{aligned}
\text{Didea}_{c^s}(s,x) &= \log \mathrm{p}(\tau_0 = 0 | x, z, C^s = c^s) = \log \mathrm{p}(\tau_0 = 0, x, z, C^s = c^s) - \log \mathrm{p}(x, z, C^s = c^s) \\
&= \log \mathrm{p}(\tau_0 = 0, x, z, c^s) - \log \sum_{\tau=-L}^{L} \mathrm{p}(\tau_0 = \tau, x, z, c^s) \\
&= \log \mathrm{p}(\tau_0 = 0, x, z, c^s) - \log \sum_{\tau=-L}^{L} \mathrm{p}(\tau_0 = \tau) \mathrm{p}(x, z_\tau | \tau_0 = \tau, c^s) \\
&= \log \mathrm{p}(\tau_0 = 0, x, z, c^s) - \log \frac{1}{|\tau_0|} \sum_{\tau=-L}^{L} \mathrm{p}(x, z_\tau | \tau_0 = \tau, c^s). \qquad (6.3)
\end{aligned}$$

As previously mentioned, $\text{Didea}_{c^s}(s,x)$ is a foreground score minus a background score, where the background score consists of averaging over $|\tau_0|$ shifted versions of the foreground score, much like SEQUEST's XCorr. Thus, Didea may be thought of as a probabilistic analogue of SEQUEST. As this scoring function is a posterior probability, this quantity is readily computable, given Didea's definition in Figure 6.3, by sum-product inference in GMTK, supported by the junction tree algorithm (an example command line of computing Didea's scoring function in GMTK is given in Appendix A.1). By defining Didea in GMTK, this allows one to alter the GM without requiring significant code redevelopment and the subsequent debugging time which would follow. This point is especially pertinent as we next discuss a significant alteration to the behavior of Didea (Section 6.2.3).

6.2.3 Integrating over precursor charge states

MS/MS spectra are commonly assigned multiple charge states, resulting from uncertainty of the precursor charges during data collection. Multiply charged observed spectra result in multiple peptide candidate sets, partitioned by charge, in which case we have a top-ranking PSM for each charge candidate set. Since only a single peptide generated the observed spectrum, it is the job of the scoring algorithm to decide which peptide this is amongst the top-ranking charge PSMs. This job is made difficult by the fact that charge 2 theoretical

spectra contain only singly charged peaks whereas charge 3 theoretical spectra contain both singly and doubly charged peaks, so that a charge 3 theoretical spectrum is denser with theoretical peaks than its charge 2 counterpart. Thus, scoring functions which are monotonically increasing in the number of theoretical peaks will assign a higher score to a charge 3 PSM than a charge 2 PSM.

In order to make PSM scores comparable between differently charged candidate sets, Didea integrates over the precursor charge states, $\{2, 3\}$, regardless of the observed precursor charge. That is to say, Didea assumes that $C^s \in \{2, 3\}$ and marginalizes over this variable. Didea's scoring function to score charge varying spectra is thus

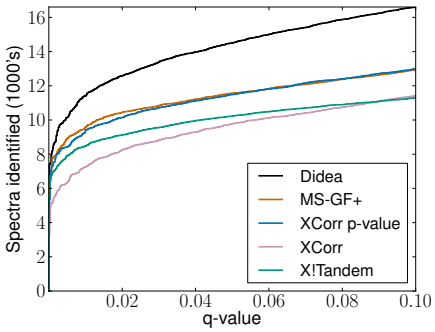
$$\begin{aligned}\text{Didea}(s, x) &= \log \mathrm{p}(\tau_0 = 0 | x, z) = \log \sum_{c^s=2}^{3} \mathrm{p}(\tau_0 = 0, x, z, c^s) - \log \sum_{c^s=2}^{3} \mathrm{p}(x, z, c^s) \\ &= \log \sum_{c^s=2}^{3} \mathrm{p}(\tau_0 = 0, x, z, c^s) - \log \frac{1}{|\tau_0|} \sum_{c^s=2}^{3} \sum_{\tau=-L}^{L} \mathrm{p}(x, z_\tau | \tau_0 = \tau, c^s). \qquad (6.4)\end{aligned}$$

6.2.4 Didea Search Accuracy

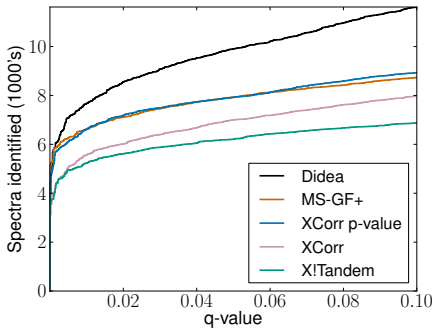
Didea is benchmarked against MS-GF+, XCorr, XCorr p-value, and X!Tandem using eight datasets collected using low-resolution MS1 and MS2 scans. Dataset descriptions may be found in Section 4.1 and search algorithm parameters are discussed in Section 4.2. As seen in Figure 6.4, Didea significantly improves performance relative to all methods across all datasets.

6.2.5 Didea's Relationship to XCorr

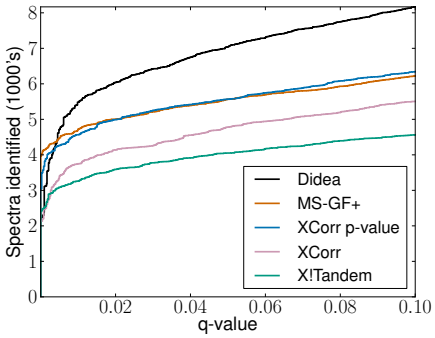
In [63], the design of Didea's log-posterior scoring function drew inspiration from XCorr. This is most obviously seen in Equations 6.3 and 6.4, where, as previously noted, Didea's scoring function is similar in form to XCorr; both consist of foreground scores, which only consider the non-shifted observed spectrum, minus background scores, which consist of an average over shifted versions of the observed spectrum. In [63], much of Didea's performance is attributed to its log-posterior's similarities to XCorr. Herein, we show the explicit relationship between



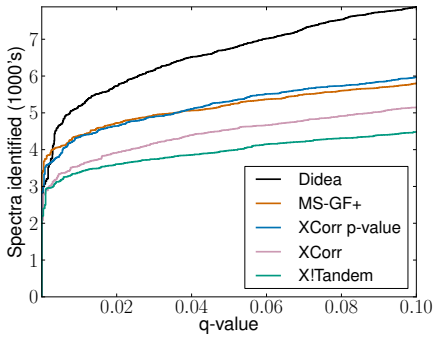
(a) Worm-1



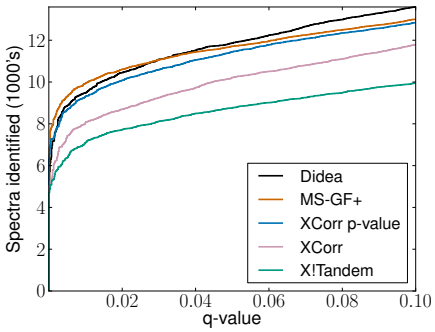
(b) Worm-2



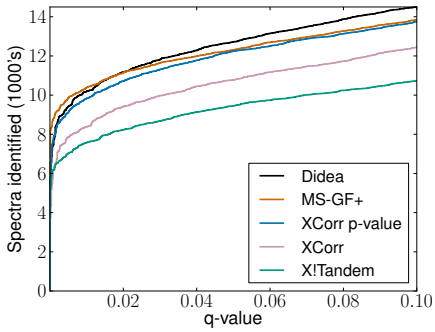
(c) Worm-3



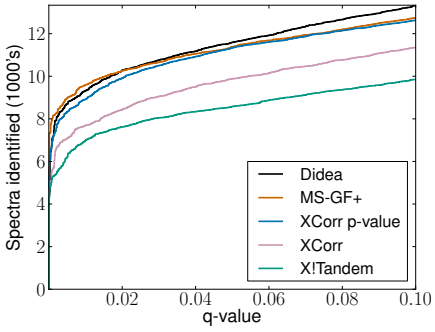
(d) Worm-4



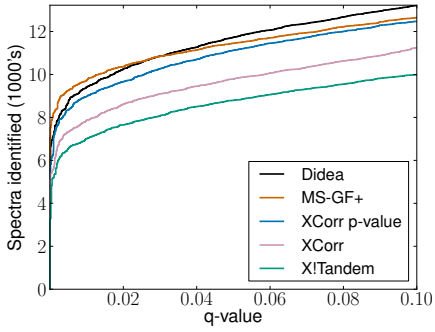
(e) Yeast-1



(f) Yeast-2



(g) Yeast-3



(h) Yeast-4

Figure 6.4: Didea performance for eight datasets.

these two scoring functions, proving that Didea's scoring function is upper bounded by XCorr. From Equation 6.3, we have

$$\begin{aligned}\text{Didea}(s,x) &= \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \log \sum_{\tau\in\mathcal{T}} \mathrm{p}(\tau_0=\tau)\mathrm{p}(x,z_\tau|\tau_0=\tau) \\ &= \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \log \mathbb{E}_\tau[\mathrm{p}(x,z_\tau|\tau_0=\tau)]\end{aligned}$$

where marginalization over τ is equivalent to an expectation. The negative of the logarithm is convex, so by Jensen's inequality, we have

$$\text{Didea}(s,x) \leq \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \mathbb{E}_\tau[\log \mathrm{p}(x,z_\tau|\tau_0=\tau)]. \tag{6.5}$$

Let $g(s,x) = \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \mathbb{E}_\tau[\log \mathrm{p}(x,z_\tau|\tau_0=\tau)]$. We thus have

$$\begin{aligned}g(s,x) &= \log \mathrm{p}(x,z|\tau_0=0)\frac{1}{|\mathcal{T}|} - \mathbb{E}_\tau[\log \mathrm{p}(x,z_\tau|\tau_0=\tau)] \\ &= -\log|\mathcal{T}| + (b+y)^T z' - \sum_{\tau\in\mathcal{T}} \mathrm{p}(\tau_0=0)\log\exp\left[(b+y)^T z'_\tau\right] \\ &= -\log|\mathcal{T}| + (b+y)^T z' - \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}}(b+y)^T z'_\tau. \end{aligned}\tag{6.6}$$

Note that if we write $u = b + y$, we have

$$\begin{aligned}g(s,x) &= -\log|\mathcal{T}| + u^T z' - \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}} u^T z'_\tau \\ &= -\log|\mathcal{T}| + u^T(z' - \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}} z'_\tau)\end{aligned}$$

so that

$$\text{Didea}(s,x) \leq -\log|\mathcal{T}| + u^T(z' - \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}} z'_\tau). \tag{6.7}$$

Thus, not only can Didea be thought of as a probabilistic analogue of SEQUEST, but Didea's scoring function is upper bounded by a function equvialent to XCorr plus a constant. This upper bound holds with equality when the random variable in the expectation is deterministic (i.e., $\mathrm{p}(x,z_\tau|\tau_0=\tau)$ is the same for all $\tau\in\mathcal{T}$). Further upper and lower bounds for Didea's scoring function are provided in Appendix B.

6.2.6 Implications of Didea's upper bound

Didea's log-posterior, Didea(s, x), is highly nonlinear and does not lend itself to a simple recursion with which to apply dynamic programming in order to compute p-value quantities over the universe of candidate peptides (as done in MS-GF+ and XCorr p-values, described in detail in Section 3.1.4). However, $g(s, x)$ is linear and we may compute p-values for it efficiently exactly as is done in XCorr (as described by Algorithm 1).

Further study of Didea's upper bound, for which we may efficiently compute exact p-values of, may allow us to approximate p-values for the original Didea scoring function itself. A variational approach may be taken, wherein a distribution of interest (which is typically infeasible to compute in practice) is approximated via a simpler auxiliary function which may be tractably computed. Critical to this strategy is that the auxiliary function bound the distribution of interest, so that minimizing the auxiliary function (when it is an upper bound) with respect to user defined free parameters leads to an accurate estimate of the distribution of interest. Bounds on Didea's scoring function provide future avenues to approximate p-values for this complicated scoring function. Specifically, we may parameterize $g(s, x)$ and minimize with respect to these parameters to gain a linear function which well approximates Didea's scoring function and for which p-values may be efficiently computed, as described in Section 3.1.4.

6.3 DBN for Rapid Identification of Peptides (DRIP)

The temporal sequence modeled in DRIP is the observed spectrum, where each frame of the model corresponds to a single observed peak. Denote the number of frames as n^s and let $\widetilde{n}^s = n^s - 1$ and t be an aribtrary frame $0 \leq t \leq \widetilde{n}^s$. The theoretical spectrum, whose variables are grouped together in the top light blue portion of Figure 6.5, is hidden and traversed from left to right as follows. K_{t-1} denotes the index of the current theoretical peak in frame $t - 1$ while δ_t is a multinomial random random variable which denotes the number

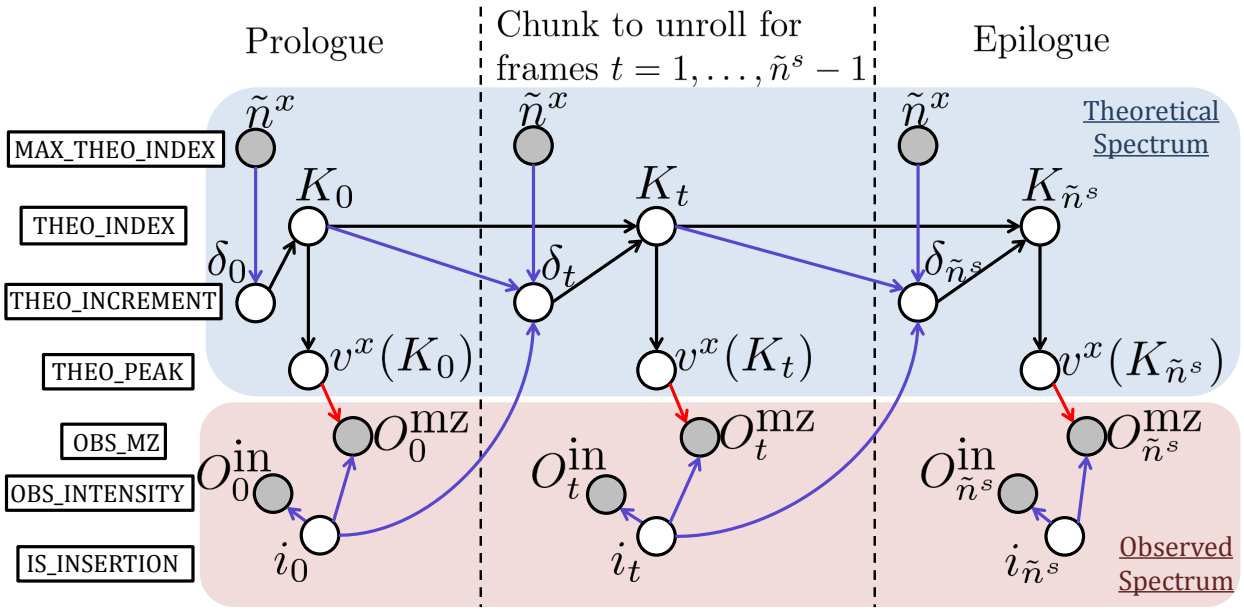


Figure 6.5: Template of DRIP. Shaded nodes represent observed variables while unshaded nodes represent hidden variables. Black edges correspond to deterministic functions of parent variables, red edges correspond to conditional Gaussian distributions, and blue edges represent switching parent functionality where the parent nodes are known as switching parents such that the parents and thus conditional distributions of their children may change given the value of switching parents.

of theoretical peaks we move down in the subsequent frame, i.e.,

$$\mathrm{p}(K_0 = \delta_0 | \delta_0) = 1, \qquad \mathrm{p}(K_t = K_{t-1} + \delta_t | K_{t-1}, \delta_t) = 1,\, t > 0. \tag{6.8}$$

Correspondingly, $v^x(K_t)$ is the K_tth theoretical peak. Note that a deletion thus occurs when $\delta_0 > 0$ and $\delta_t > 1$ for $t > 0$, i.e., the *hypotheses* such that one or more theoretical peaks are not accessed, where a hypothesis is an assignment of all random variables in the graph. The number of deletions occuring in a single frame is then δ_0 for the prologue and $(\delta_t - 1)\mathbf{1}\{\delta_t > 1\}$ for all subsequent frames. In order to discourage the use of deletions unless absolutely necessary, the distribution over δ_t is constrained to be monotone decreasing. The parents of δ_t, $\widetilde{n}^x$ and K_{t-1}, constrain it from being larger than the number of remaining theoretical peaks left to traverse, i.e.,

$$\mathrm{p}(\delta_0 > \widetilde{n}^x | \widetilde{n}^x) = 0, \qquad \mathrm{p}(\delta_0 > \widetilde{n}^x - K_{t-1} | \widetilde{n}^x, K_{t-1})\, t > 0.$$

6.3.1 Scoring observed peaks

Variables modeling the observed spectrum are grouped together in the bottom light red portion of Figure 6.5. The variables O_t^{mz} and O_t^{in} are the observed m/z and intensity values, respectively, of the tth observed peak, so that as t increases we move further down the m/z axis of the observed spectrum. The m/z observation is scored by a Gaussian centered near the theoretical peak accessed in a frame and the intensity observation is scored using a Gaussian with mean greater than or equal to the most intense peak value. In this manner, DRIP aligns the theoretical and observed spectra by penalizing observed peaks far from theoretical peaks and penalizing peaks with intensity less than unity. The variable i_t, parent to both O_t^{mz} and O_t^{in}, is a Bernoulli random variable denoting whether the tth observed peak is an insertion, $i_t = 1$, or not. When $i_t = 0$, the observations are scored as

$$\mathrm{p}(O_t^{\mathrm{mz}} | v^x(K_t), i_t = 0) \sim \mathcal{N}(\mu^{\mathrm{mz}}(v^x(K_t)), \sigma^2), \qquad \mathrm{p}(O_t^{\mathrm{in}} | \mu^{\mathrm{in}}, i_t = 1) \sim \mathcal{N}(\mu^{\mathrm{in}}, \bar{\sigma}^2),$$

where μ^{in} and $\bar{\sigma}^2$ are the mean and variance of the Gaussian used to score peak intensities, σ^2 is the variance of the Gaussian used to score m/z observations, and μ^{mz} is a vector of

means such that an arbitrary theoretical peak j scores m/z observations using $\mathcal{N}(\mu^{\text{mz}}(j), \sigma^2)$. The theoretical peaks serve as indices into a vector of Gaussians, each such Gaussian having equal variance and centered at a unique m/z position (illustrated in Figure 6.6). Thus, when describing a theoretical peak as scoring an observed peak, we are referring to scoring using the Gaussian accessed by the theoretical peak. To avoid confusion, we refer to the Gaussian accessed by a theoretical peak as the *theoretical Gaussian peak*.

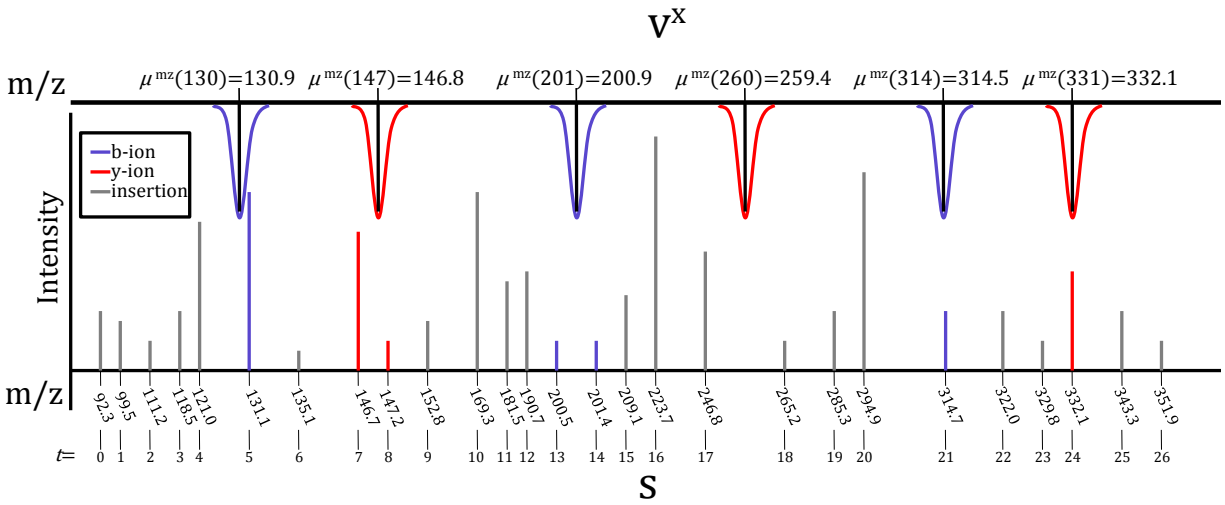


Figure 6.6: Example spectra alignment in DRIP given an observed spectrum s, $c^s = 2$, $x = \text{EALK}$, the theoretical Gaussian peaks of v^x and the hypothesis is listed in Table 6.1. Note that t is the frame number, all Gaussians have equal variance, and $v^x(3), v^x(4)$ are deletions.

When $i_t = 1$, a constant penalty is returned rather than scoring the observations with the currently considered Gaussians dictated by K_t, since scores may become arbitrarily bad by allowing Gaussians to score m/z observations far from their means and this would lead to a large dynamic range of scores. These constant penalties, denoted as a_{mz} and a_{in} for the m/z observation and intensity observation, respectively, are carefully selected to impose a tradeoff between scoring observed peaks which are close to theoretical peaks or are high intensity. Furthermore, so as to enforce that some observations be scored rather than inserted,

Table 6.1: Random variable hypothesis for alignment displayed in Figure 6.6. Recall the deterministic relationships $K_0 = \delta_0$, $K_t = K_{t-1} + \delta_t$ for $t > 0$, and given the theoretical peak index K_t we have the theoretical peak $v^x(K_t)$. For instance, from the theoretical spectrum of $x = \text{EALK}, c(x) = 2$ in Figure 2.4, we have $K_6 = K_5 + \delta_6$ and $v^x(K_6) = v^x(1) = 147$.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
δ_t	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0
K_t	0	0	0	0	0	0	1	1	1	2	2	2	2	2	2	4	4	4	4	4	4	4	5	5	5	5	5
i_t	1	1	1	1	1	0	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	1	1

i_t enforces its child δ_{t+1} to be zero in a frame following an insertion. Because δ_t and i_t are hidden, DRIP considers all possible *alignments*, i.e., all possible combinations of insertions and theoretical peaks scoring observed peaks. The Viterbi path, i.e., the alignment which maximizes the log-likelihood of all the random variables, is used to score a peptide as well as determine which observed peaks are most likely insertions and which theoretical peaks are most likely deletions.

Note that the version of DRIP used in this work (Figure 6.5) is simplified relative to the initially described DRIP model [28]. In particular, the previously described version of DRIP required two estimated quantities, corresponding to the maximum allowable numbers of insertions and deletions. These constraints were enforced via two chains of variables, which kept track of the number of utilized insertions and deletions in an alignment, counting down in subsequent frames. The current, simplified model automatically determines these two quantities on the basis of the deletion and insertion probabilities as well as the insertion penalties. This simplification comes at no detriment to performance (Figure 6.7) or running time (experiment not shown). Max-product inference in a GM, even one as complicated as DRIP, to compute the Viterbi path is supported in GMTK, making the exploration of model variations possible without requiring significant code revision and debugging (an example

command line to compute the Viterbi path of DRIP in GMTK is given in Appendix A.2).

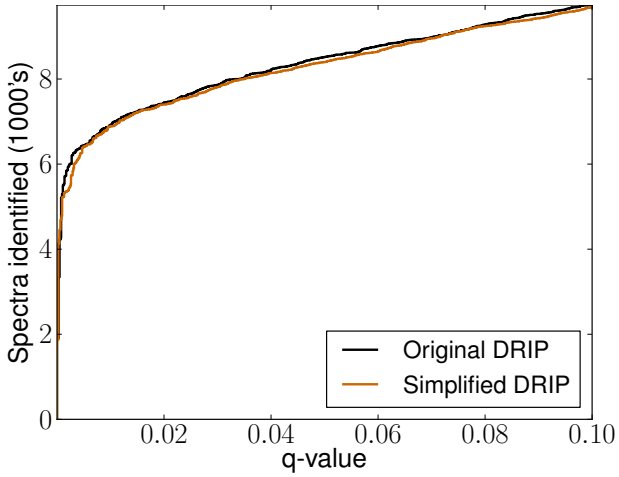


Figure 6.7: Performance of DRIP with two Markov chains limiting the number of allowable deletions and insertions (called "Original DRIP") versus simplified DRIP relying on carefully selected insertion and deletion penalties without explicitly limiting the number of deletions and insertions (called "Simplified DRIP").

6.3.2 DRIP scoring function

Peptides are scored by their optimal alignment using their per-frame log-Viterbi Score,

$$\psi(s,x) = \frac{1}{n^s} \max_{\delta_t, i_t, \forall t} \log \mathrm{p}(s|x) = \frac{1}{n^s} \log \mathrm{p}^*(s|x). \tag{6.9}$$

Dividing by the number of frames allows comparability of PSMs from different spectra. In order to further analyze the scoring function, assume inference has been completed and we have computed the Viterbi path, using * to denote a variable's Viterbi value. Let $\lambda = \sum_{t=0}^{\widetilde{n}^s} i_t^*$ denote the number of used insertions and note that $\mathrm{p}(i_t = 0) = \mathrm{p}(i_0 = 0)$. DRIP's score is

then

$$\log \mathrm{p}^*(s|x) = \sum_{t=0}^{\widetilde{n}^s} [\log \mathrm{p}(\delta_t^*) + \mathbf{1}\{i_t^* = 0\}(\log f(O_t^{\mathrm{mz}}|\mu^{\mathrm{mz}}(v^x(K_t^*)), \sigma^2) + \log f(O_t^{\mathrm{in}}|\mu^{\mathrm{in}}, \sigma^2))] \tag{6.10}$$

$$+ \lambda[\log(a_{\mathrm{mz}} a_{\mathrm{in}}) + 3 \log \mathrm{p}(i_0 = 1)] + 3(n^s - \lambda) \log \mathrm{p}(i_0 = 0),$$

where, as before, $f(z|\mu, \bar{\sigma}^2)$ is the scalar Gaussian with mean μ and variance $\bar{\sigma}^2$, evaluated at $z \in \mathbb{R}$. The learned model variances are such that $\sigma^2 << \bar{\sigma}^2$, i.e., there is far more uncertainty in intensity measurements than m/z measurements. Thus, it is easy to see that when a peptide does not align well with the observed spectrum (i.e., many observed peaks are far from the closest theoretical peak), the $\log f(O_t^{\mathrm{mz}}|\mu^{\mathrm{mz}}(v^x(K_t^*)), \sigma^2)$ term severely penalizes the score. Furthermore, this score decreases exponentially as the distance between the observed peak and theoretical peak mean increases. This also means that peptides which arbitrarily match intense peaks will still receive poor scores if they do not align well.

6.3.3 DRIP observed spectrum preprocessing

As with all other search algorithms, we score database peptides of at most a fixed maximum length, specified prior to run time. Practical values of the maximum peptide length (the maximum peptide length considered for all results presented is 50) mean that the number of observed peaks is typically an order of magnitude larger than the number of theoretical peaks for any scored peptide. Furthermore, most of these peaks are noise peaks [60], and as such we filter all but the most l intense peaks, where in practice, $l = 300$.

After filtering peaks, the observed spectrum is renormalized as in SEQUEST [20], the steps of which are as follows. Firstly, all observed peak intensity values are replaced with their square root. Secondly, the observed spectrum is partitioned into 10 equally spaced regions along the m/z axis and all peaks in a region are normalized to some globally maximum intensity, which in our case is 1. Steps one and two greatly decrease the high observed intensity variance, and step two helps ensure that scoring is not severely biased by many

large peaks lying arbitrarily close to one another. Lastly, any peak with intensity less than 1/20 of the most intense peak is filtered.

6.3.4 Calibrating DRIP with respect to charge

Observed fragmentation spectra with low-resolution precursor data often have indeterminate charge states. For such spectra, all candidates are scored and ranked assuming all charge states, with the highest scoring peptide amongst all charges returned. This approach requires that scores among differently charged spectra be comparable to one another. In DRIP, however, this is not the case. Because the number of theoretical peaks essentially doubles when moving from charge 2+ to charge 3+, higher charged PSMs which have denser theoretical spectra and thus much better scores than lower charged PSMs, rendering differently charged PSMs incomparable.

In order to alleviate this problem, we recalibrate scores on a per charge basis, projecting differently charged PSM scores to roughly the same range. The procedure consists of first generating a secondary set of decoy peptides disjoint from the target peptide set and primary decoy peptide sets. Let $\mathcal{D}^t$ be the set of target peptides, $\mathcal{D}^d$ be the set of decoy peptides, and $\mathcal{D}^{dd}$ be our new set of decoy peptides, such that $\mathcal{D}^{dd} \cap (\mathcal{D}^t \cup \mathcal{D}^d) = \emptyset$. Next, all spectra and charge states are searched, returning top ranking sets of PSMs $\mathcal{X}^t \subseteq \mathcal{D}^t$, $\mathcal{X}^d \subseteq \mathcal{D}^d$, and $\mathcal{X}^{dd} \subseteq \mathcal{D}^{dd}$. We then partition $\mathcal{X}^{dd}$ based on charge. For each charge c and corresponding partition, $\mathcal{Z}^{dd}$, we rank all PSMs based on score. Let $\mathcal{Z}^t \subseteq \mathcal{X}^t$ be all PSMs of charge c, for which we use their scores to linearly interpolate their normalized ranks in $\mathcal{Z}^{dd}$, using these interpolated normalized ranks as their new calibrated scores. In order to recalibrate scores greater than those found in $\mathcal{Z}^{dd}$, we use the 99.99th percentile score and maximal score for linear interpolation. Once this procedure is done, a majority of recalibrated scores for PSMs in $\mathcal{Z}^t$ and $\mathcal{Z}^d$ will lie between $[0, 1]$ and a few barely above unity (this is largely true for the targets, since a set of well expressed and identifiable target peptides typically outscores all decoys), thus greatly decreasing the dynamic range of scores. With these recalibrated scores, we may easily take the top ranking PSM of an observed spectrum amongst differently

charged PSMs, without any loss in overall accuracy.

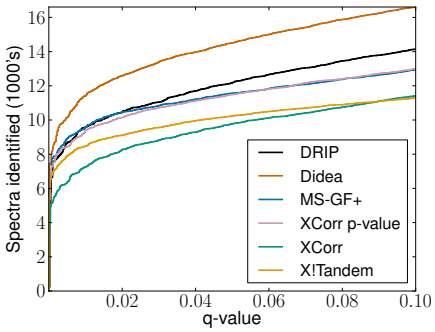
6.3.5 DRIP Search Accuracy

DRIP is benchmarked against Didea, MS-GF+, XCorr, XCorr p-value, and X!Tandem using eight datasets collected using low-resolution MS1 and MS2 scans. Further details regarding search settings, datasets, and how competitors were run may be found in Section 6.2.4. As seen in Figure 6.8, DRIP is outperformed only by Didea on all Worm datasets and is comparable to the calibrated tide p-value for all Yeast datasets. Note that the presented DRIP results employ the generatively trained model (Section 8.1.1).)

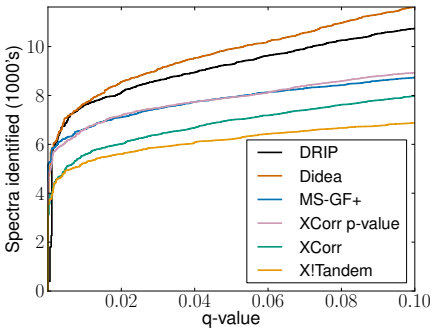
6.3.6 DRIP for high-resolution fragment ions

In order to take advantage of the greater certainty in fragment ion locations afforded by high-high data (discussed in Section 2.3), the theoretical peak Gaussians DRIP utilizes change in two significant way. Firstly, with the increased certainty of m/z measurements, Gaussians are centered directly at fragment ion locations. Previously, for high-low and low-data spectra, fragment ions were integerized and used to access one of a finite number of Gaussians (the means of which were learned). Secondly, the variance of all Gaussians is significantly smaller than for low-resolution fragment ions. For the results presented in Figure 6.9, the variance for all Gaussians is set such that 99.99% of the Gaussian mass lies within a 0.05 Th region.

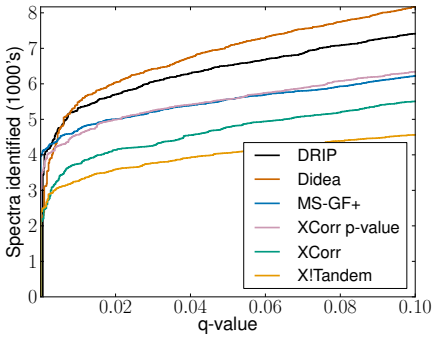
In Figure 6.9, DRIP is benchmarked against MS-GF+, XCorr, XCorr p-value, and X!Tandem, where all but XCorr p-value are run with settings reflecting high-high data (the recursion utilized to compute XCorr p-values is not currently defined to utilize high-resolution fragments). The presented malaria (*Plasmodium falciparum*) sample was digested using Lys-C, labeled with an isobaric tandem mass tag (TMT) relabeling agent, and collected using high-resolution precursor scans and high-resolution fragment ions. The data set consists of 12,594 spectra with charges ranging from 2+ through 6+. Searches were run using a 50 ppm tolerance for selecting candidate peptides, a 0.03 Th fragment mass tolerance, a fixed



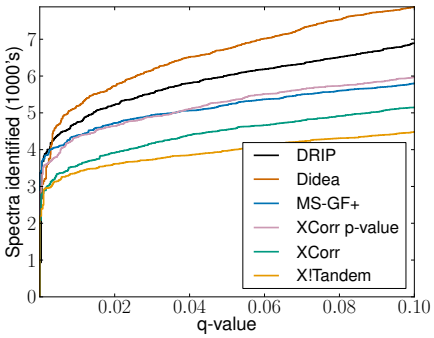
(a) Worm-1



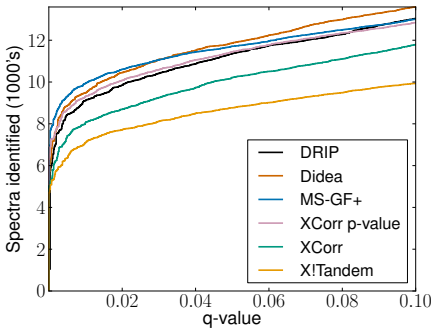
(b) Worm-2



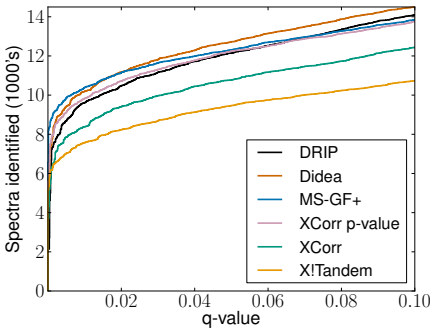
(c) Worm-3



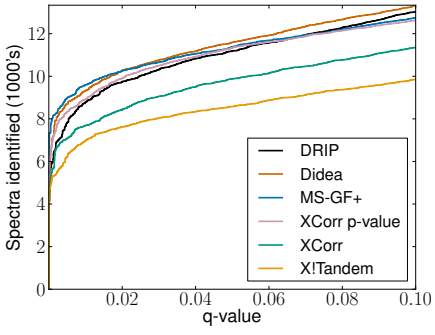
(d) Worm-4



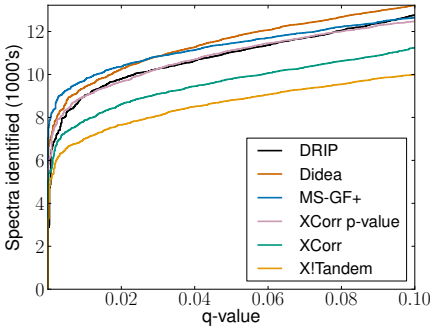
(e) Yeast-1



(f) Yeast-2



(g) Yeast-3



(h) Yeast-4

Figure 6.8: Drip performance for eight datasets.

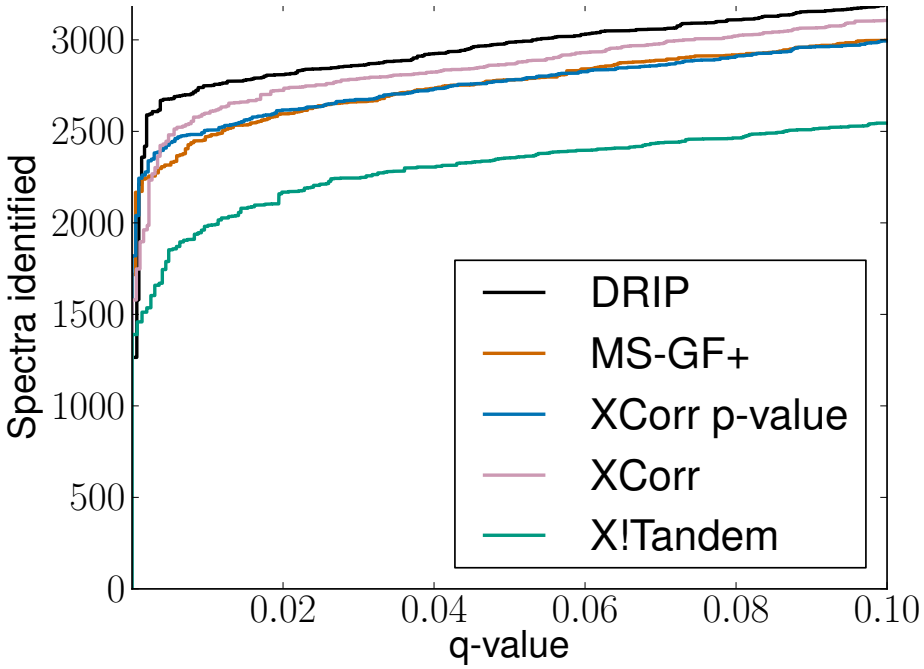


Figure 6.9: DRIP tested over a high-high Malaria dataset.

carbamidomethyl modification, a fixed TMT labeling modification of lysine and N-terminal amino acids. Further details may be found in [74].

Chapter 7

FASTER IDENTIFICATION OF PEPTIDES USING GRAPHICAL MODELS

We now show how graphical models may be used to significantly reduce MS/MS search time by representing all peptide candidates per observed spectrum under the same data structure. In general, this is made difficult by the restriction that we would only like to consider those candidates from the database of interest. For instance, for the log-linear model described in Section 6.1, which we described in the context of XCorr, although we could make the theoretical peak nodes (u_t) hidden, it is a difficult to enforce that only valid theoretical spectra be considered, since the difference between theoretical peaks is arbitrary due to the theoretical spectrum being a union between b- and y-ions. Furthermore, even if theoretical peak units were not an issue, constraining the model to only consider peptides from the database is also a difficult task without further machinery. We thus turn to *trellises*, which allow us to construct the state-space of a graphical model to exactly the set of theoretical peaks for an arbitrary set of peptides.

A *trellis* is a graph over strings, the nodes of which are substrings and the edges of which are characters from the alphabet of interest. Paths through the trellis correspond to strings of interest. When constructing the trellis, the goal is to share edges amongst common substrings, affording a compact representation of a large number of strings (a potentially exponential number of strings in sub-exponential space). An example trellis is illustrated in Figure 7.1(b) encoding the phrases in Figure 7.1(a). The green node denotes the source node, from which all paths begin, and the red node denotes the sink node, where all paths must end. As shown in Figure 7.1(b), subpaths in the trellis correspond to common subsequences amongst the original sequences. Due to this commonality, trellises are commonly used used

to share redundant computation in graphical models [35]. General trellises may contain sequences outside the original set to be modeled (e.g., "Bananas are really bad" is not in the input set of phrases in Figure 7.1(a), but are a valid path in the trellis in Figure 7.1(b)). We thus only allow *exact trellises* (i.e., trellises only containing the original input set of sequences, illustrated in Figure 7.1(c) for the phrases in Figure 7.1(a)) during construction (further detailed in [72]). Trellises have been used to great success in speech recognition and natural language processing, enabling fast decoding of speech signals [69, 51] and rendering the discriminative training for large scale speech recognition tractable [59].

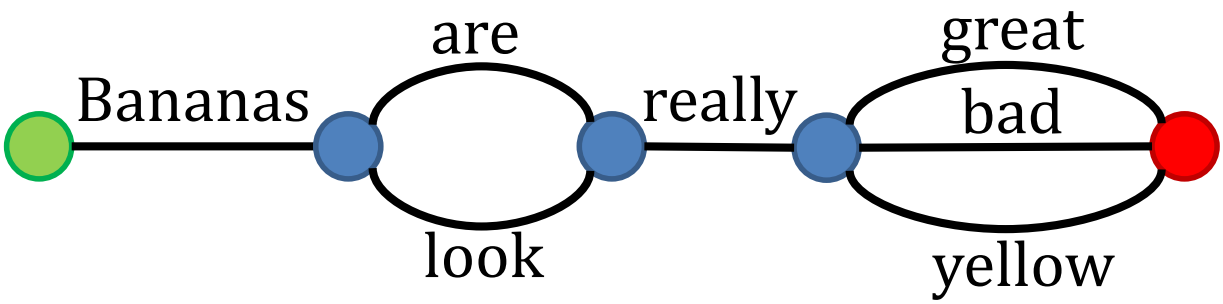
For our MS/MS database search, trellises allow us to construct GM state spaces comprised of the set of theoretical spectra for the candidate peptides within a precursor mass tolerance. With all the peptides candidates under the same roof, we may thus calculate the Viterbi path to determine the top scoring PSM for linear models, such as XCorr, and DRIP. Redundant computation is thus performed only once for the common sequences of theoretical peaks amongst the candidate peptides. Note that, in the case of log-linear models (Section 6.1), the state space is exactly the set of all candidate peptide theoretical spectra, whereas for DRIP (Section 6.3), the state space is the set of all alignments between all the candidate peptides theoretical spectra and the given observed spectrum. Owing to the use of Viterbi inference, we use approximate inference, which we now describe in detail, to significantly improve search runtime.

7.0.7 Approximate inference via pruning

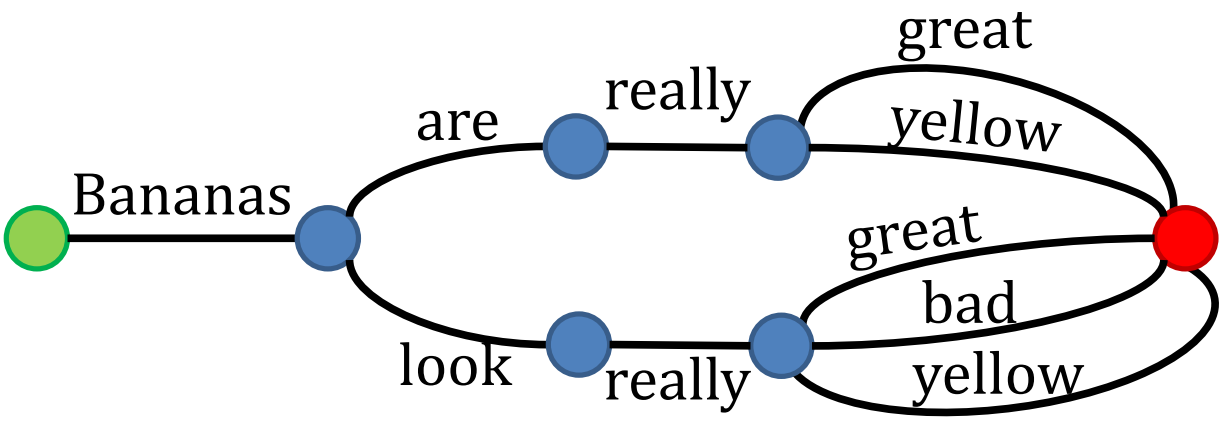
Due to the temporal nature of DBNs, it is often desirable to perform approximate inference when exact inference is too expensive. Such might be the case when the graph in each frame is complicated, there are a large number of frames, or there are many low probability configurations of random variables disjoint from the Viterbi path. In approximate inference, we sacrifice guaranteeing that BP converges to the BP fixed point (i.e., the solution were we to run normal BP over a junction tree) to speed up inference time. One particular set of approximate inference algorithms which have been successful for DBNs are beam pruning

Bananas are really great
Bananas are really yellow
Bananas look really great
Bananas look really bad
Bananas look really yellow

(a) Five phrases describing bananas.



(b) General trellis containing five phrases regarding bananas.



(c) Exact trellis over five phrases regarding bananas.

Figure 7.1: Phrases describing bananas compressed using trellises.

methods [56].

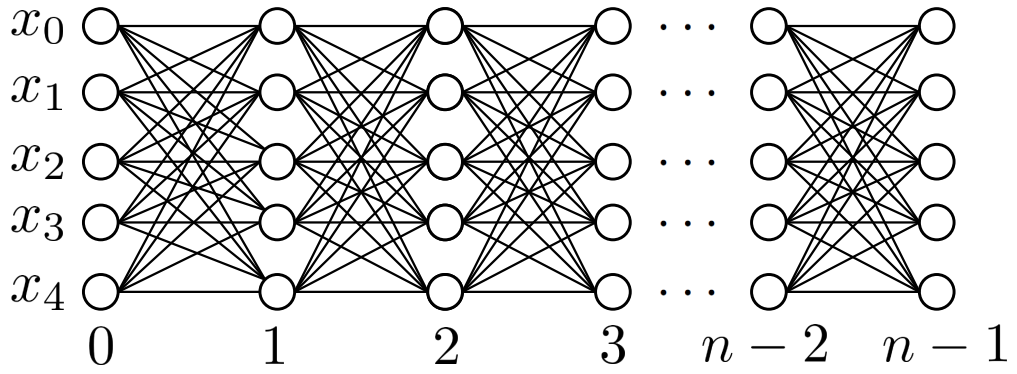


Figure 7.2: Lattice graph of HMM state space with five hidden states x_0, x_1, x_2, x_3, x_4.

In k-beam pruning (also called histogram pruning), assuming a beam width of $k \in \mathbb{N}$, only the top k most probable states in a frame are allowed to persist. Such pruning has a profound effect on the state space of the DBN. To see this, consider the HMM from Figure 5.6(a), with five hidden states $X_i \in \{x_0, x_1, x_2, x_3, x_4\}$. The state space of such an HMM is often compactly represented via a *lattice* graph, which, for our case, is depicted in Figure 7.2 (this type of graph is typically called a trellis, but we refer to it as a lattice herein to avoid confusion).

In Figure 7.2, each node denotes a state the hidden layer may take on in a frame, where frames are denoted by the number below the graph. Each edge in the lattice denotes a possible transition from one hidden state to another in the subsequent frame. A path through the lattice thus denotes a particular instantiation of the random variables in the HMM. Such an instantiation of random variables is called a *hypothesis*. For instance, considering an HMM of length 3, one hypothesis would be x_0, x_1, x_2, another would be x_2, x_3, x_4, and so on, where each hypothesis corresponds to a path through the lattice. For our HMM, we have $|X_i|^n = 5^n$ possible hypotheses. As one would expect, when $|X_i|$ and/or n are large, the number of hypotheses grows large very quickly.

By utilizing a beam width of k in k-beam pruning, we ensure only the top k hypotheses up to the current frame survive, thus pruning a potentially exponential number of hypotheses. This approximate inference strategy and related variants have been especially effective in applications utilizing Viterbi inference, where the only hypothesis of interest is the most probable one. However, care must be taken so that k is sufficiently large to ensure that the most probable hypothesis is not pruned early on. For instance, if we set a conservative beam width where x_0 was pruned in the very first frame, then all hypotheses beginning with x_0 would subsequently be pruned. If it were the case that the most probable sequence began with x_0, we would never recover the Viterbi path. Thus, there is a tradeoff between inference time and setting k sufficiently large. Luckily for our MS/MS GMs of interest, the Viterbi path is robust so that we may prune many low-probability hypotheses early on.

7.1 Timing results

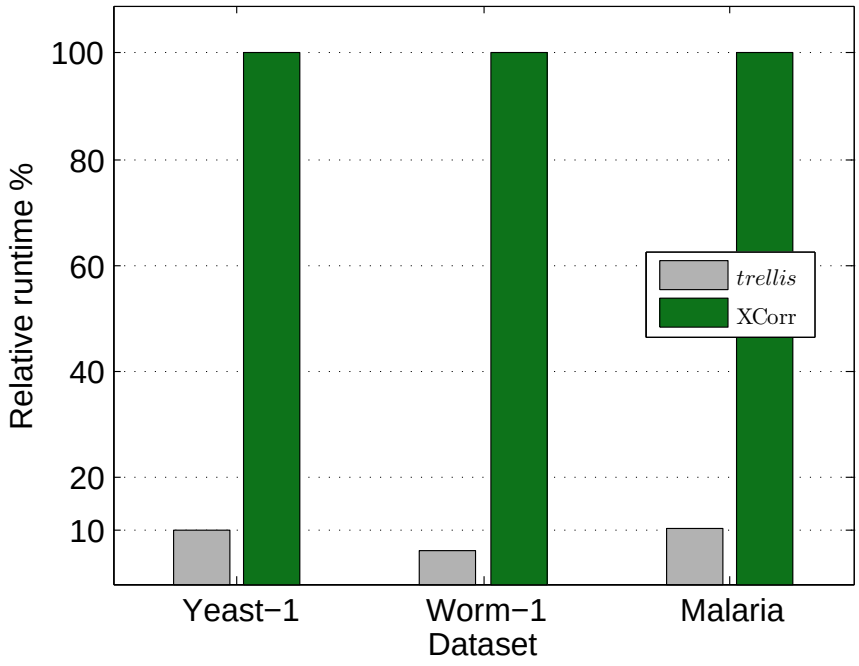
Instead of pruning away hypotheses of each data instance, beam pruning with trellises prunes hypotheses of all data instances together, thereby achieving higher efficiency. For example, the hypotheses of the original DRIP model only consist of a single peptide's alignments between an observed spectrum so that, when using beam pruning, poor alignments are pruned away early on. With trellises, beam pruning enables a competition among all peptide hypotheses, where peptide candidates which align poorly with the observed spectrum are pruned away early, so we end up scoring only a subset of the candidates. Trellises, max-product inference, and various types of beam pruning are supported in GMTK, facilitating the exploration of inference options to quickly test and time GMs (an example command line for max-product inference with k-beam pruning may be found in Appendix A.3).

Using the same graphical model inference engine (GMTK) for all models, we report relative timing numbers to clearly demonstrate the algorithmic speed-ups afforded by trellises. Experiments were carried out on a 3.40GHz CPU with 16G memory. For each experiment, the lowest CPU time out of three runs is recorded, and we report the relative CPU time of methods using trellises to those without. We test over three different datasets, two low-low

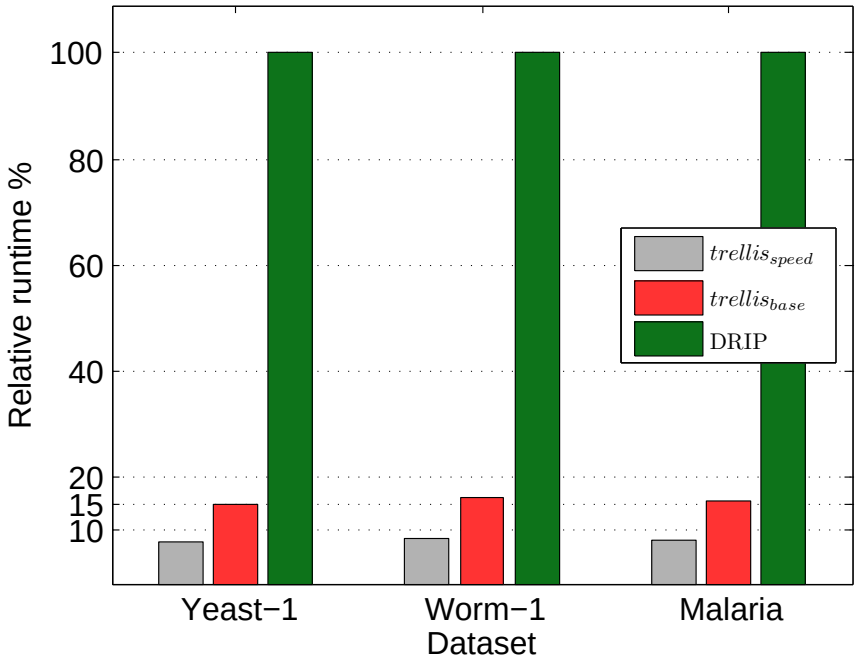
datasets (Yeast-1 and Worm-1) and one high-high dataset (Malaria). We randomly select and search 200 spectra each of the three datasets, using the precursor mass tolerance settings listed in Table 4.1. Under the control of using the same graphical model engine, the relative timing results reflect the ratio of computation required among different methods. Figure 7.3 displays the relative timing for DRIP and XCorr without trellises (the green bars, which are the baseline for timing) versus with (raw relative runtime numbers may be found in Appendix E).

For the XCorr mixture model, a fixed beam width k was used for all frames. As we can see, XCorr with trellis runs significantly faster than without, achieving 10 and 16.2 fold speed-ups on the low-low datasets Yeast-1 and Worm-1, respectively, and a 9.5 fold speed-up on the high-high Malaria dataset. We test DRIP trellis with two beam pruning strategies. The $trellis_{base}$ pruning uses k-beams that are dynamic across time frames, with wider beams for the early part and narrower beams later on. The $trellis_{speed}$ pruning also uses a dynamic beam width but with a narrow beam followed by pruning strategy all hypotheses whose score falls below some fraction of the currently top scoring hypothesis in the current frame. Timing tests show that DRIP runs 12.9, 11.6, and 12.4 times faster using trellises than without trellises for Yeast-1, Worm-1, and Malaria, respectively.

Thus, trellises combined with beam pruning methods significantly improve MS/MS search time, both for low-resolution and high-resolution data. This combination significantly speeds up both simple models, such as the XCorr mixture model, and the much more complicated DRIP model. We note that the log-linear mixture model may be easily adjusted to model any other linear MS/MS scoring function, of which there are many, such as X!Tandem [14], Morpheus [73], and the base scores of MS-GF+ [44] and OMSSA [24]; changing the observed spectrum preprocessing will produce these other score functions, without a change of the graphical model itself. Thus, in principle, the XCorr speed-ups displayed herein extend to these other linear scoring functions. We further note that the algorithmic speed-up afforded by using trellises may potentially be combined with previous work on improving XCorr runtime, which focused on constant-factor improvements [16].



(a) XCorr trellis timing results.



(b) DRIP trellis timing results.

Figure 7.3: DRIP and XCorr trellis timing results.

Chapter 8

LEARNING GRAPHICAL MODELS FOR PEPTIDE IDENTIFICATION

8.1 Training DRIP

All DRIP Gaussian means and variances have been effectively learned using a high-confidence set of PSMs used in [47]. Note that this adds a great deal of modeling power to DRIP, allowing for adaptability of the expected m/z location of theoretical peaks. In MS/MS data, the m/z measurements may be offset by machine error [19] or contaminants present in the complex sample. Learning the Gaussian means allows DRIP to account for and adapt to these trends. Furthermore, the learned means themselves may be of interest to experts in the field. GMTK supports both learning frameworks utilized for DRIP (examples command lines may be found in Appendix A).

8.1.1 Generative training

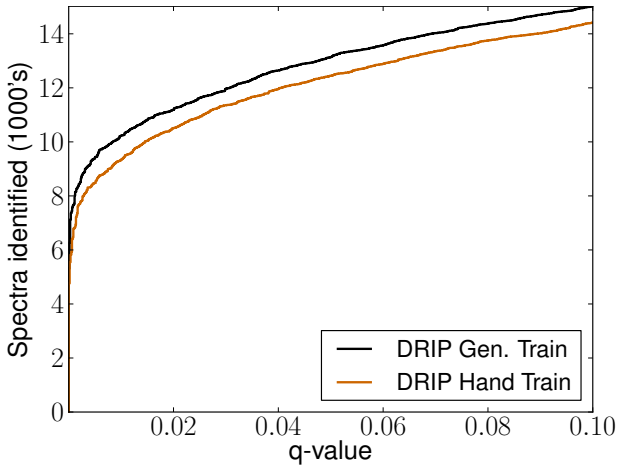
For the overall training procedure, assume that we have a collection, C, of N i.i.d. pairs (s^i, x^i), where s^i is an observed spectrum and x^i the corresponding PSM we believe to have generated s^i. Let θ be the set of parameters for which we would like to learn, in our case, DRIP's Gaussian parameters. For generative training, we wish to maximize DRIP's likelihood with respect to the parameters to be learned, i.e.,

$$\theta^* = \operatorname*{argmax}_{\theta} \sum_{i=1}^{N} \log p(s^i | x^i, \theta),$$

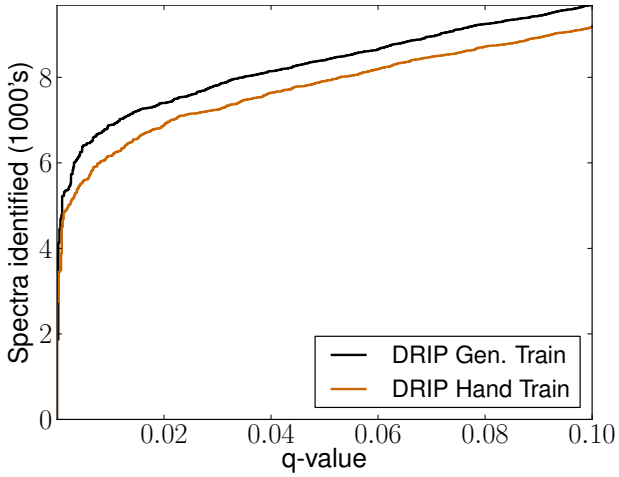
where θ^* is determined via expectation-maximization (EM) [15].

Utilizing a high-quality set of PSMs collected in [47], generatively learning DRIP's Gaussian parameters significantly improves performance relative to hand-tuned parameters

(Figure 8.1). The hand-tuned parameters were chosen to represent the fixed binning strategies of other existing methods such that each Gaussian mean is positioned in the center of a unit interval along the m/z axis.



(a) Worm-1, charge 2+



(b) Yeast-1, charge 2+

Figure 8.1: Generatively training DRIP significantly improves performance.

8.1.2 Discriminative training

A much more difficult training paradigm is that of discriminative training, wherein we do not simply wish to maximize the likelihood under a set of parameters, but we would also like to simultaneously minimize a parameterized distribution defined over a set of alternative hypotheses. In our case, this alternative set of hypotheses consists of all candidate peptides within the precursor mass tolerance not equal to x^i, i.e., all incorrect explanations of s^i. More formally, our discriminative training criterion is that of Maximum Mutual Information Estimation (MMIE); defining the set of candidate peptides for s^i within precursor mass tolerance w as $\mathcal{C}^i = D(O^{\text{mz}}, c^s, \mathcal{D}, w)$ and denoting the set of all training spectra and high-confidence PSMs as $\mathcal{S}$ and $\mathcal{X}$, respectively, the function we would like to maximize with respect to θ is then

$$\begin{aligned} I_\theta(\mathcal{S};\mathcal{X}) =& \mathbb{E}\log\frac{p(s^i,x^i|\theta)}{p(s^i|\theta)p(x^i|\theta)} = \sum_{s^i,\mathcal{C}^i} p(s^i,\mathcal{C}^i|\theta)\log\frac{p(s^i,x^i|\theta)}{p(s^i|\theta)p(x^i|\theta)} \\ =& \sum_{s^i,\mathcal{C}^i} p(s^i,\mathcal{C}^i|\theta)\log\frac{p(s^i|x^i,\theta)p(x^i|\theta)}{p(s^i|\theta)p(x^i|\theta)} = \sum_{s^i,\mathcal{C}^i} p(s^i,\mathcal{C}^i|\theta)\log\frac{p(s^i|x^i,\theta)}{p(s^i|\theta)} \\ =& \sum_{s^i,\mathcal{C}^i} p(s^i,\mathcal{C}^i|\theta)\log\frac{p(s^i|x^i,\theta)}{\sum_{x\in\mathcal{C}^i}p(s^i,x|\theta)}. \end{aligned} \tag{8.1}$$

Note that, while mutual infomation is typically defined over random variables, here we've defined $I_\theta(\mathcal{S};\mathcal{X})$ over sets.

We approximate this objective using the quantity $\tilde{I}_\theta(\mathcal{S};\mathcal{X}) = \frac{1}{N}\sum_{i=1}^{N}\log\frac{p(s^i|x^i,\theta)}{\sum_{x\in\mathcal{C}^i}p(s^i,x|\theta)}$, which converges to the quantity in Equation 8.1 for large N by the i.i.d. assumption and the weak law of large numbers. Our objective then is to find θ^* such that

$$\begin{aligned} \theta^* = \operatorname*{argmax}_\theta \tilde{I}_\theta(\mathcal{S};\mathcal{X}) =& \operatorname*{argmax}_\theta \frac{1}{N}\sum_{i=1}^{N}\log\frac{p(s^i|x^i,\theta)}{\sum_{x\in\mathcal{C}^i}p(s^i,x|\theta)} \\ =& \operatorname*{argmax}_\theta \frac{1}{N}\sum_{i=1}^{N}(\log p(s^i|x^i,\theta) - \log\sum_{x\in\mathcal{C}^i}p(s^i,x|\theta)), \end{aligned} \tag{8.2}$$

where, for obvious reasons, we refer to $M_n(s^i,x^i) = \log p(s^i|x^i,\theta)$ as the *numerator model* and $M_d(x^i) = \log\sum_{x\in\mathcal{C}^i}p(s^i,x|\theta)$ as the *denominator model*. Note that the numerator model

is our objective function for generative training. In general, the sum over possible peptide candidates in the denominator model makes the discriminative training objective difficult to compute. To efficiently perform the computation in the denominator model, we utilize trellises (Section 7). The optimization in Equation 8.2 is performed via stochastic gradient ascent, detailed in Section 8.1.2.

Stochastic Gradient Ascent

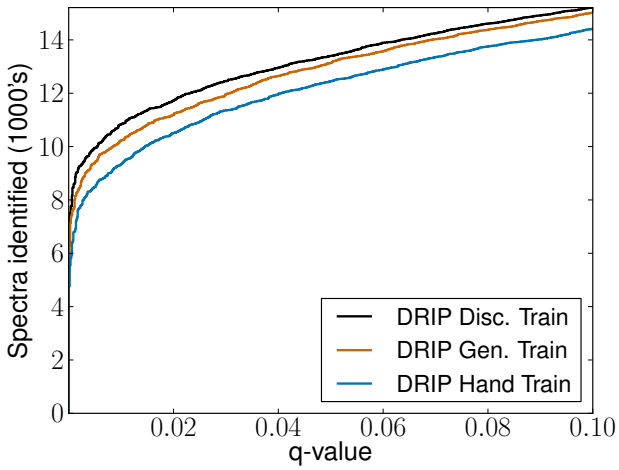
In stochastic gradient ascent, we calculate the gradient of the objective function with regard to a single training instance,

$$\nabla_\theta \tilde{I}_\theta(s^i; x^i) = \nabla_\theta M_n(s^i|x^i\theta) - \nabla_\theta M_d(s^i|\theta), \tag{8.3}$$

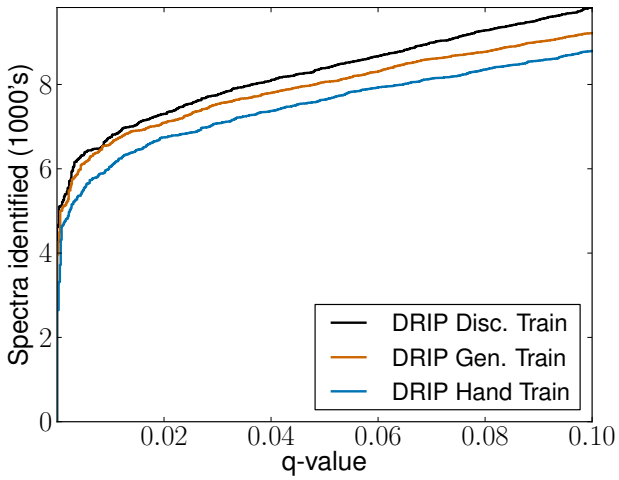
where the gradients of M_n and M_d are vectors referred to as *Fisher scores*. Correspondingly, we update the parameters θ using the previous parameters plus a damped version of Equation 8.3, iterating this process until convergence. The overall algorithm is detailed in Algorithm 2. In practice, we begin the algorithm by initializing θ_0 to a good initial value, i.e., the generatively learned parameters, and the learning rate η_j is updated with $\eta_{j+1} = (\sqrt{j})^{-1}$. Intuitively, the gradients move in the direction maximizing the difference between the numerator and denominator models, encouraging improvement for the numerator while discriminating against the incorrect labels in the denominator. Our experimental results show that discriminative training improves performance relative to generative training at low q-values (Figure 8.2), arguably the most difficult region to improve upon.

8.2 Training Didea

We show how hyperparameters for Didea's scoring function may be efficiently learned. This section assumes knowledge of Didea's scoring function as detailed in Section 6.2. Consider an observed spectrum $s \in \mathcal{R}^{2l}$ and binned, quantized, and pre-preprocessed observed spectrum $z \in \mathcal{R}^{\bar{o}}$, where $\bar{o}$ is the maximum observable integer of the mass spectrometer. Didea's preprocessing consists of replacing each peak intensity with its normalized rank amongst all



(a) Worm-1, charge 2+



(b) Yeast-1, charge 2+

Figure 8.2: Discriminatively training DRIP improves performance over generative training.

Algorithm 2 Discriminative Training via Stochastic Gradient Ascent

Initialize θ_0, η_0. Let $j = 0$.
while True **do**
    $\theta_{j+1} := \theta_j$; $j := j + 1$;
    Update learning rate η_j;
    for $(s^i, x^i) \in C$ **do**
        $\theta_j := \theta_j + \eta_j(\nabla_\theta M_n(s^i)|x^i\theta) - \nabla_\theta M_d(s^i|\theta))$;
    end for
    if $||\frac{\theta_j - \theta_{j-1}}{\theta_{j-1}}|| < \epsilon$ **then**
        break;
    end if
end while

peak intensities, followed by reweighting processed intensities using a complicated, tailor-made function. The function used in [63] to reweight rank-normalized peak intensities is

$$f_\lambda(r) = 1 - \lambda e^{-\lambda} + \lambda e^{-\lambda(1-r)} \tag{8.4}$$

so that, for rank-normalized intensity β falling in bin j, we have $z(j) = f_\lambda(\beta)$.

Now, consider a charge 2+ peptide $x = x_0 x_1 \dots x_{n-1}$ with b-ions $b_1, b_2, \dots, b_{n-1}$ and y-ions $y_1, y_2, \dots, y_{n-1}$. Define $\tilde{n} = n - 1$. Optimizing λ in $f_\lambda(i)$ is difficult as $f_\lambda(i)$ is non-convex and Didea's resulting posterior scoring function is both complicated and non-convex utilizing this reweighting function, necessating a grid-search in [63] to find a well-performing λ. In practice, a grid-search requires an entire database search over a training dataset per iteration. Each iteration requires many hours on a single machine, so that convergence may require days to complete in practice. Furthermore, this grid-search utilizes absolute ranking as a measure of performance and is thus semi-supervised. We make Didea's learning framework much more efficient without any cost to performance by reformulating the log-posterior as a concave function of λ which is optimized in seconds in practice, as opposed

to days using grid-search on a single machine. Furthermore, this new framework is easily extendable to other parameterizations of Didea's log-posterior where the set of parameters to be learned is expanded, but remain concave in the function of interest so that learning remains tractable. We show that one such natural extension results in improved identification accuracy. Considerations for parameterization of scoring functions similar in form to Didea's log-posterior, and for which parameter estimation remains tractable even for large numbers of parameters, are discussed at length in Appendix D, where the tractable parameterizations which we now utilize may be found in Appendix D.2.2.

8.2.1 Concave reparameterization of Didea's scoring function

For a vector $v \in \mathbf{R}^{\bar{o}}$ and shift $\tau \in [-M, M]$, let v_τ denote the vector v whose elements have all been shifted τ units. Assume we utilize the reweighting function $f'_\lambda(i) = e^{\lambda i}$, so that Didea's log-posterior scoring function is then

$$\begin{aligned}
\log p(s, x|\tau_0 = \tau, \lambda) &= \log \prod_{i=1}^{\tilde{n}} e^{\lambda s_\tau(b_i)} e^{\lambda s_\tau(y_i)} \\
&= \log \prod_{i=1}^{\tilde{n}} e^{\lambda(s_\tau(b_i) + s_\tau(y_i))} \\
&= \log e^{\lambda[\sum_{i=}^{\tilde{n}}(s_\tau(b_i) + s_\tau(y_i))]} \\
&= \log e^{\lambda(b^T s_\tau + y^T s_\tau)} \\
&= \log e^{\lambda(b+y)^T s_\tau} \\
&= \lambda(b + y)^T s_\tau,
\end{aligned}$$

where b and y are boolean vectors of length $\bar{o}$ such that for the ith b- and y-ions of x, b_i and y_i, we have $b(b_i) = 1$ and $y(y_i) = 1$ and zero otherwise.

Didea's scoring function is then

$$\begin{aligned}\log p(\tau_0 = 0|s, x, \lambda) &= \log e^{\lambda(b+y)^T s} - \log \sum_{\tau=-M}^{M} e^{\lambda(b+y)^T s_\tau} \\ &= -\log \sum_{\tau=-M}^{M} e^{\lambda[(b+y)^T s_\tau - (b+y)^T s)]} \\ &= -\log \sum_{\tau=-M}^{M} e^{\lambda(b+y)^T (s_\tau - s)} \end{aligned} \tag{8.5}$$

Note that Equation 8.5 is concave in the quantity $\lambda(b+y)^T(s_\tau - s)$. Defining $z \in \mathbb{R}^{|\mathcal{T}|}$ s.t. $z(i) = \lambda(b+y)^T(s_i - s)$, we can see that based on the ranges of b, y, and s, $-2 \leq z(i) \leq 2$. Assume that we have a set $\mathcal{D}$ of N i.i.d. observed spectrum and peptide pairs, $\mathcal{D} = \{(s^1, x^1), (s^2, x^2), \ldots, (s^N, x^N)\}$, the posterior over all i.i.d. shifts $\tau_0^1, \tau_0^2, \ldots, \tau_0^N$ is then

$$\log p(\tau_0^1 = 0, \ldots, \tau_0^N = 0|s^1, \ldots, s^N, x^1, \ldots, x^N, \lambda) = \sum_{i=1}^{N} \log p(\tau_0^i = 0|s^i, x^i, \lambda). \tag{8.6}$$

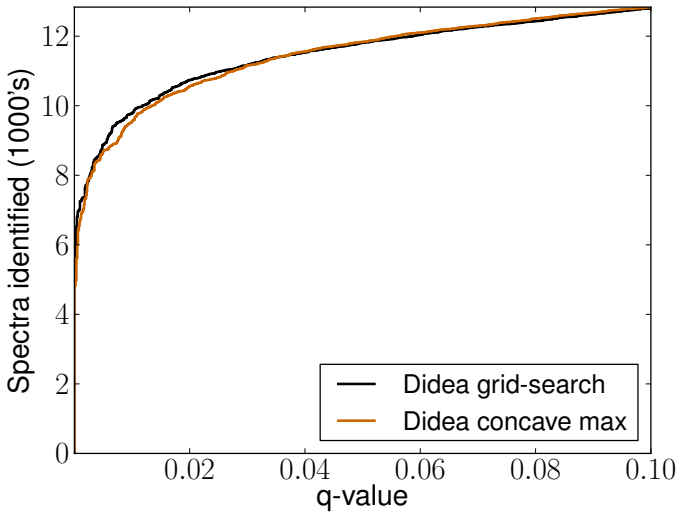
Equation 8.6 remains concave due to the concavity of each $\log p(\tau_0^i = 0|s^i, x^i, \lambda)$. Defining $z^i(\tau) = (b_{x^i} + y_{x^i})^T(s^{i,\tau} - s^i)$ and $g_{\mathcal{D}}(\lambda) = \sum_{i=1}^{N} \log p(\tau_0^i = 0|s^i, x^i, \lambda)$, our optimization objective is thus

$$\max_{\lambda} \quad g_{\mathcal{D}}(\lambda) = \max_{\lambda} \sum_{i=1}^{N} \log p(\tau_0^i = 0|s^i, x^i, \lambda) = \max_{\lambda} -\sum_{i=1}^{N} \log \sum_{\tau_0^i=-M}^{M} e^{\lambda z^i(\tau_0^i)}. \tag{8.7}$$

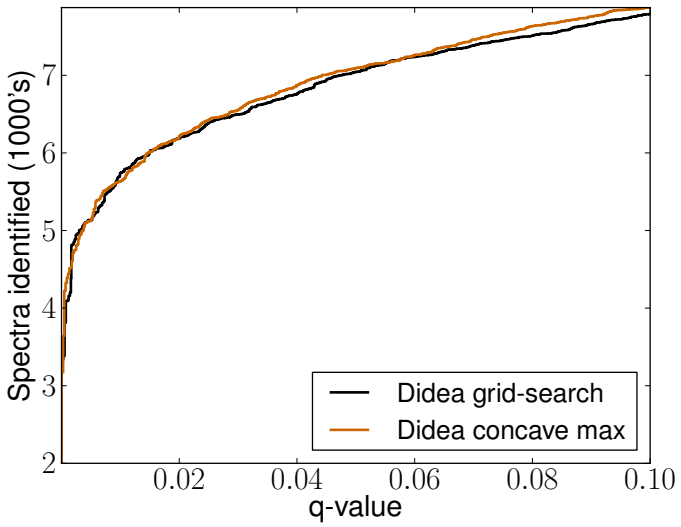
Differentiating with respect to λ, we have

$$\frac{d}{d\lambda} g_{\mathcal{D}}(\lambda) = -\sum_{i=1}^{N} \frac{1}{\sum_{\tau_0^i} e^{\lambda z^i(\tau_0^i)}} \sum_{\tau_0^i} z^i(\tau_0^i) e^{\lambda z^i(\tau_0^i)}. \tag{8.8}$$

We may efficiently learn the global optimum, $\lambda^* = \text{argmax}_\lambda \sum_{i=1}^{N} \log p(\tau_0^i = 0|s^i, x^i, \lambda)$ using Equation 8.8, and a steepest descent method. We utilize stochastic gradient ascent (SGA), defined in Algorithm 2. Training may also be sped up by precaching each $z^i(\tau_0^i)$ in Equation 8.8. We note that this is overall scheme is an instance of efficient discriminative training of a generative model, by virtue of Didea's conditional log-likelihood scoring function.



(a) Worm-1, charge 2+



(b) Worm-2, charge 2+

Figure 8.3: Absolute ranking for Didea under $f_\lambda(\cdot)$ optimized using a grid-search ("Didea grid-search") and Didea under $f'_\lambda(\cdot)$ optimized using a SGA ("Didea concave max") for Worm-01, charge 2+. **Relative ranking**:

Dataset	Didea grid-search	Didea concave max
Worm-01	0.836869	**0.841460**
Worm-02	0.729957	**0.732219**

Utilizing the set of PSMs collected in [47] (the same data used to train DRIP in Section 8.1), training converges in seconds utilizing a single core of an Intel Core i7-330 3.4GHz processor with 16GB of memory). This is in stark contrast to the grid-search performed in [63], which requires days to complete utilizing the same compute environment as each iteration of the grid-search requires a full target/decoy database search and thus many hours to complete. Furthermore, there is no significant performance loss utilizing λ^* and the new formulation of Didea's scoring function utilizing $f'_\lambda(\cdot)$ compared to utilizing the complicated reweighting function $f_\lambda(\cdot)$ from [63], as seen in Figure 8.3. Finally, we note that this new framework leads to improved relative ranking.

8.2.2 Concave reparameterization of Didea's scoring function for improved search accuracy

We now define a natural extension to the concave reparameterization of Didea's scoring function. In this extension, we will greatly increase the number of parameters to be learned but, by remaining in the realm of concavity, efficiently learn all parameters. This leads to fast parameter learning and improved identification accuracy, especially at low q-values.

For weight vector $\boldsymbol{\lambda} \in \mathcal{R}^{2M+1}$, our objective function to be optimized is now

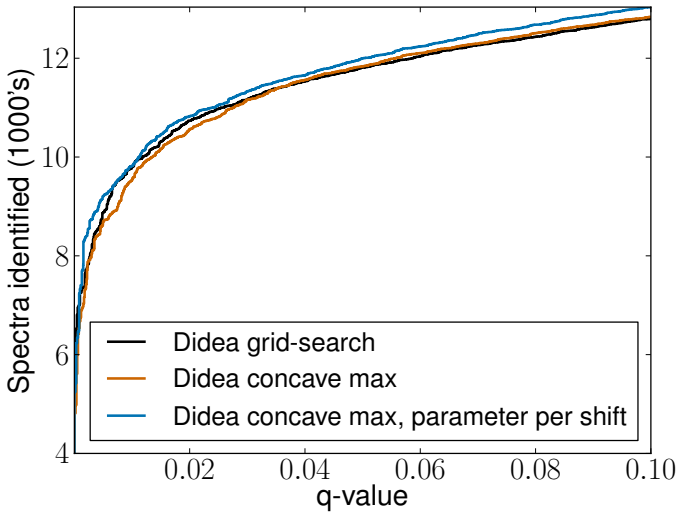
$$\max_{\boldsymbol{\lambda}} \quad g_{\mathcal{D}}(\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}} \sum_{i=1}^{N} \log p(\tau_0^i = 0 | s^i, x^i, \boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}} - \sum_{i=1}^{N} \log \sum_{\tau_0^i = -M}^{M} e^{\lambda_{\tau_0^i} z^i(\tau_0^i)} \tag{8.9}$$

the partial derivatives of which are f

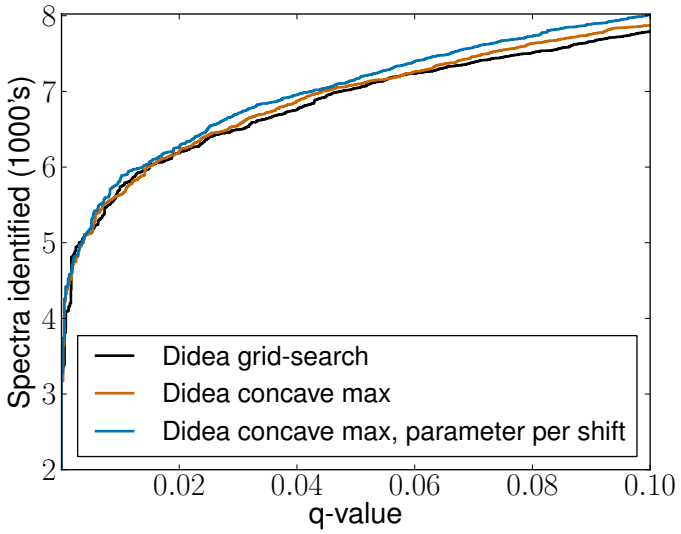
$$\frac{\delta}{\delta \lambda_i} g_{\mathcal{D}}(\boldsymbol{\lambda}) = - \sum_{i=1}^{N} \frac{1}{\sum_{\tau_0^i} e^{\lambda_{\tau_0^i} z^i(\tau_0^i)}} z^i(\tau_0^i) e^{\lambda_{\tau_0^i} z^i(\tau_0^i)}. \tag{8.10}$$

Where previously, in Equation 8.7, we learned a global weight for all osberved spectrum shifts, we now learn a specific weight for each individual shift in Equation 8.9. Equation 8.9 is concave in $\boldsymbol{\lambda}$. Utilzing the same training data and compute environment described in Section 8.2.1, Equation 8.9 converges to a global optimum in a few hours using SGA. In contrast, such training would be intractable utilizing a grid-search.

As seen in Figure 8.4, weighting each observed spectrum shift differently results in improved search accuracy relative to the previously trained Didea objective functions. Having greatly



(a) Worm-1, charge 2+



(b) Worm-2, charge 2+

Figure 8.4: Absolute ranking for Didea under $f_\lambda(\cdot)$ optimized using a grid-search ("Didea grid-search") and Didea under $f'_\lambda(\cdot)$ optimized using a SGA ("Didea concave max") for Worm-01, charge 2+. **Relative ranking:**

Dataset	Didea grid-search	Didea concave max	Didea concave max, parameter per shift
Worm-01	0.836869	0.841460	**0.841549**
Worm-02	0.729957	0.732219	**0.734037**

expanded the set of learned parameters and shown these parameters have a positive effect on performance, a natural direction is to learn a sparse $\boldsymbol{\lambda}$ where only the most "important" observed spectrum shifts remain nonzero after training. This and other feasible extensions of Didea's learning framework are discussed in Appendix D.3.

Chapter 9

FEATURE EXTRACTION FOR IMPROVED POST-PROCESSING ACCURACY USING DRIP

Scoring functions often suffer from poor *calibration*, where PSM scores from different spectra are difficult to compare to one another. Thus, although a scoring algorithm may do well ranking an observed spectrum's generating peptide above all other database peptides (the task of *relative ranking*), the scoring algorithm may perform poorly in terms of ranking target PSMs above decoy PSMs from other spectra (what we refer to as absolute ranking, defined in Chapter 4). In order to address the calibration issue, methods in the field have been developed [42, 13, 17, 38, 66] which accept as inputs PSMs and post-process these PSM scores to more easily be compared to one another. These post-processors have been shown to improve performance for methods which are already known to be well-calibrated [26, 33], further displaying their utility to identification accuracy.

The most accurate such post-processing algorithms are semi-supervised, where both target and decoy PSMs are passed as inputs, along with features regarding each PSM [17, 38, 66]. Each decoy PSM is known with certainty to not have generated its corresponding observed spectrum whereas, in general, we do not know with certainty whether a target PSM generated its observed spectrum. These partial labels are used to denote the targets and decoys as positive and negative classes, respectively, for classification.

The widely used post-processor Percolator [38] uses the target and decoy PSMs along with their features to train a support vector machine (SVM). Utilizing the learned support vectors, the target and decoy scores are adjusted such that their respective score distributions are better separated in order to improve identification accuracy. Critical to reranking performance are the features used, which typically involve the peptide charge, peptide mass, difference

Table 9.1: Default Percolator features for Crux v2.1.16567.

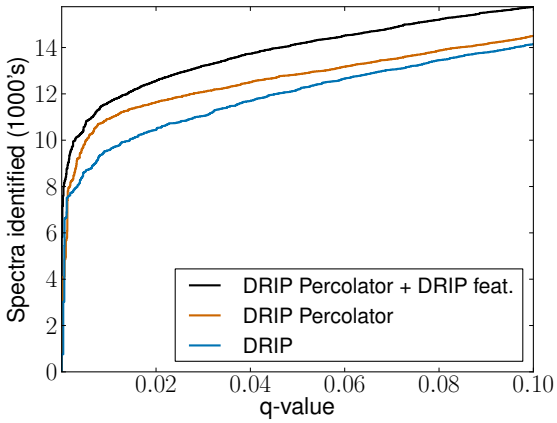
Feature	Description
`Sp`	Sp score
`lnrSp`	Natural logarithm of sp score
`deltLCn`	Difference between a PSM's XCorr and the XCorr of the last-ranked PSM, divided by the PSM's XCorr or 1, whichever is larger.
`deltCn`	Difference between a PSM's XCorr and the XCorr of the next-ranked PSM, divided by the PSM's XCorr or 1, whichever is larger.
`IonFrac`	Estimated fraction of b and y ions theoretical ions matched to the spectrum.
`Mass`	Precusor mass.
`PepLen`	Peptide length.
`Charge1`	$\mathbf{1}\{\text{PSM charge} = 1\}$
`Charge2`	$\mathbf{1}\{\text{PSM charge} = 2\}$
`Charge3`	$\mathbf{1}\{\text{PSM charge} = 3\}$
`enzN`	Is the peptide preceded by an enzymatic (tryptic) site?
`enzC`	Does the peptide have an enzymatic (tryptic) C-terminus?
`enzInt`	Number of missed internal enzymatic (tryptic) sites
`lnNumSP`	Natural logarithm of the number of database peptides within the specified precursor range
`dm`	Difference between the observed precursor and peptide mass
`absdM`	Absolute value of the difference between the observed precursor and peptide mass

between precursor and peptide mass, absolute different between precursor and peptide mass, and number of scored candidates, to name a few.

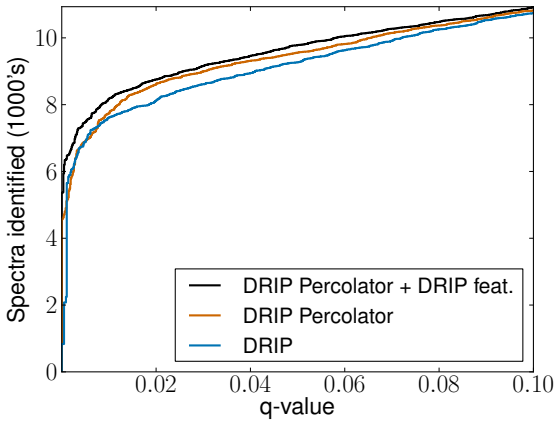
Table 9.2: DRIP derived features.

Feature	Description
`insertions`	Number of inserted observed peaks.
`deletions`	Number of deleted theoretical peaks.
`scoredPeaks`	Number of non-inserted observed peaks.
`usedTheoPeaks`	Number of non-deleted theoretical peaks.
`sumScoredIntensities`	Sum of the intensities of non-inserted observed peaks
`sumScoredMz`	Sum of the absolute differences between non-inserted observed peaks and the DRIP Gaussian mean used to score those peaks

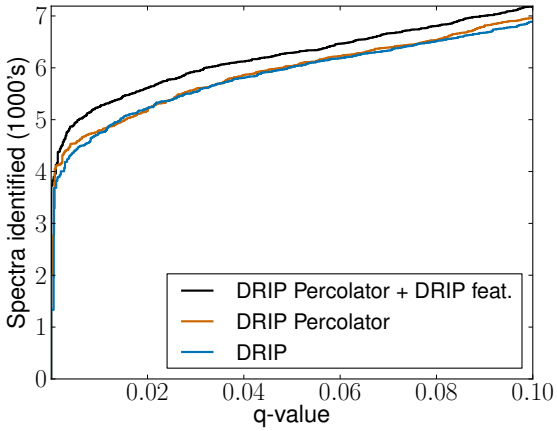
As previously noted, inference in DRIP returns much information regarding a PSM. With DRIP's Viterbi path, the sequence of insertions and deletions is known, so not only can we infer the number of insertions and deletions, but also exactly which observed peaks are insertions and which theoretical peaks are deletions. We use these quantities to add to Percolator several new features, listed in Table 9.2. The default features used for Percolator in Crux v2.1.16567 [52] and to which the DRIP derived features are added to are listed in Table 9.1. As seen in Figure 9.1, adding these DRIP derived features significantly increases performance relative to Percolator using standard features. We take this approach one step further, using DRIP to derive these features for other scoring algorithm's PSMs and the derived features to significantly improve post-processing performance. Furthermore, we show that while similar features may be easily computed using a scoring algorithm's quantization scheme, the features derived by DRIP lead to significantly better post-processing identification accuracy.



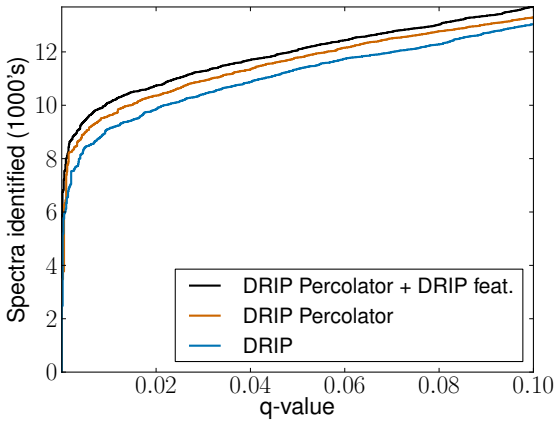
(a) Worm-1



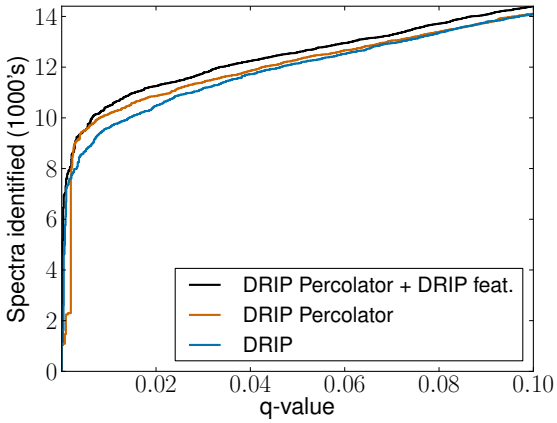
(b) Worm-2



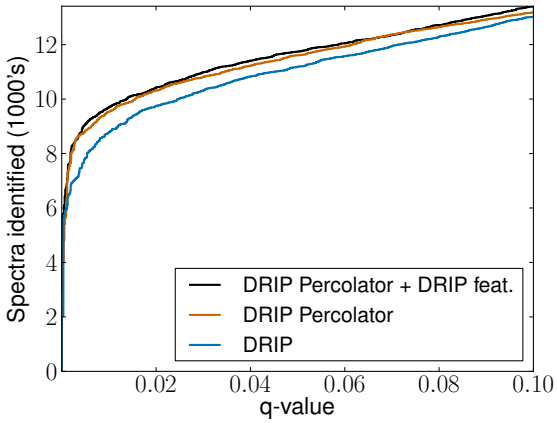
(c) Worm-3



(d) Yeast-1



(e) Yeast-2



(f) Yeast-3

Figure 9.1: DRIP utilizing the features listed in Table 9.1 as well DRIP derived features.

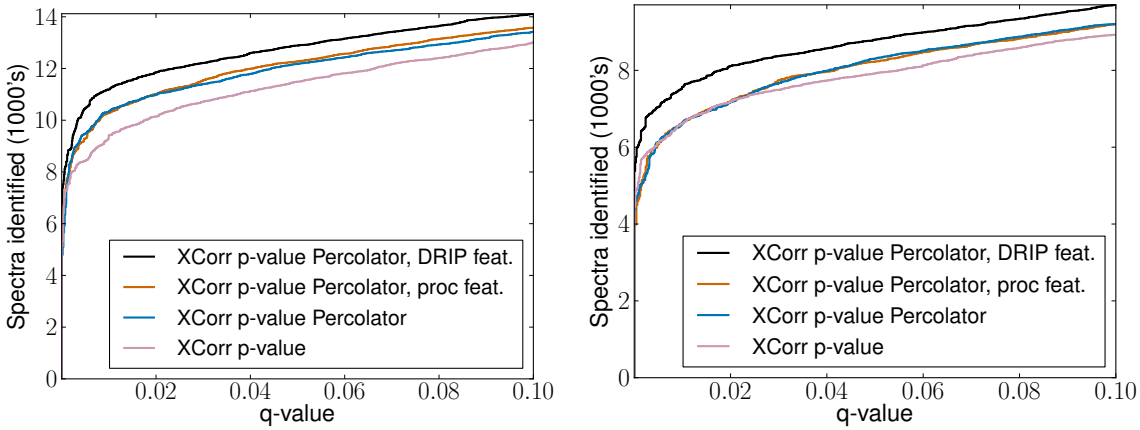
9.1 Feature extraction using DRIP

A scoring algorithm's PSMs may be fed into DRIP in order to extract the features listed in Table 9.1. These features may then be appended to the standard set of features used by a post-processor (such as Percolator) for a specific scoring algorithm. For instance, we may extract features using DRIP given the PSMs generated by XCorr p-value. Adding these features to the standard features listed in Table 9.1 (which are the same features used in conjunction with Percolator and XCorr p-values in [33]) results in the performance depicted in Figure 9.2. XCorr p-values with the DRIP extracted features significantly improves search accuracy for all datasets.

As a control to determine whether it is truly feature extraction using DRIP which leads to performance gains, we also compute features the features in Table 9.1 by quantizing the observed spectrum (as is done in many search algorithms) and adding these quantities to the standard set of XCorr Percolator features in Crux listed in Table 9.1, referred to as "XCorr p-value Percolator, proc feat." in Figure 9.2. The DRIP derived features outperform the analogous features derived using quantization, leading us to conclude that the greatest utility of these features comes from decoding a PSM's Viterbi path using DRIP.

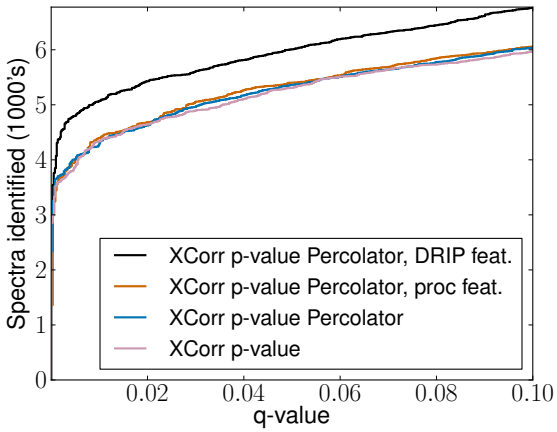
We repeat this overall experiment for all presented datasets using MS-GF+, appending features to the MS-GF+ derived features for Percolator derived in [26] (descriptions of these features are listed in Table 9.3). As was the case with XCorr p-value Percolator, the DRIP derived features helps to better classify target PSMs from decoy PSMs leading to significantly improved Percolator accuracy (Figure 9.3). Once again, the DRIP derived features significantly outperform those derived by quantizing the observed spectra (referred to as "MS-GF+ Percolator, proc feat.").

We have shown that the auxiliary information provided by DRIP has great utility, even for other scoring algorithms.

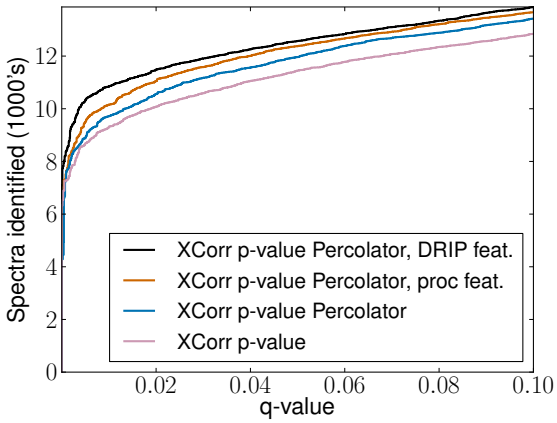


(a) Worm-1

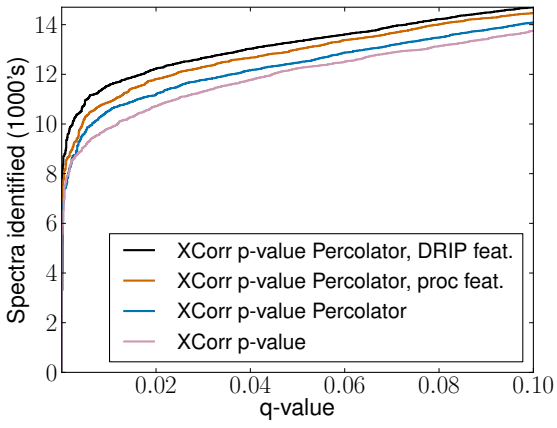
(b) Worm-2



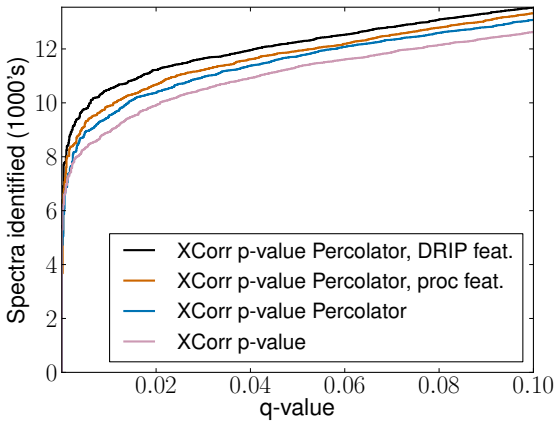
(c) Worm-3



(d) Yeast-1



(e) Yeast-2

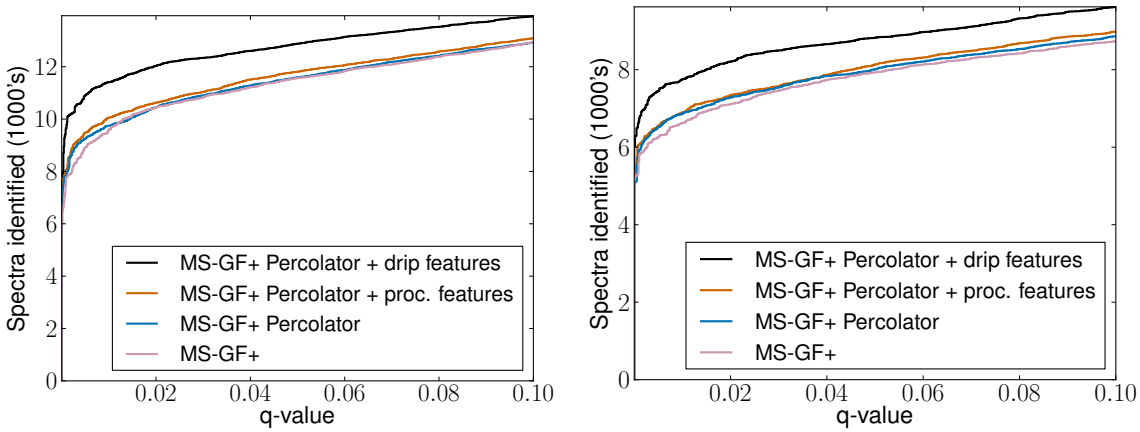


(f) Yeast-3

Figure 9.2: XCorr p-value utilizing the features listed in Table 9.1 as well DRIP derived features.

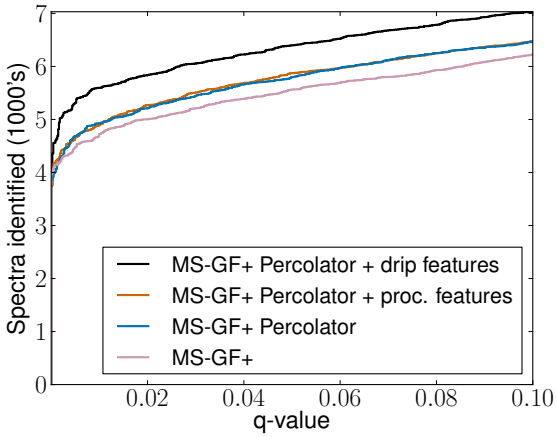
Table 9.3: MS-GF+ Percolator features, further described in [26].

Feature	Description
RawScore	MS-GF+ base score (dot-product)
DeNovoScore	Maximum possible RawScore for observed spectrum
ScoreRatio	$\frac{\text{RawScore}}{\text{DeNovoScore}}$
Energy	RawScore − DeNovoScore
lnEValue	− log MS-GF+ E value
lnSpecEValue	− log MS-GF+ Spectral E value
IsotopeError	Number of additional neutrons in peptide
LnExplainedCurrentRatio	$\log \frac{\sum\{\text{intensity of matched fragment ions}\}}{\sum\{\text{intensity of all fragment ions}\}}$
LnNtermIonCurrentRatio	$\log \frac{\sum\{\text{intensity of matched N-terminal fragments}\}}{\sum\{\text{intensity of all fragment ions}\}}$
LnCtermIonCurrentRatio	$\log \frac{\sum\{\text{intensity of matched C-terminal fragments}\}}{\sum\{\text{intensity of all fragment ions}\}}$
LnMs2IonCurrent	$\log \sum\{\text{intensity of all fragment ions}\}$
Mass	Peptide mass
Length	Peptide length
dM	Theoretical mass − experimental mass
absdM	|Theoretical mass − experimental mass|
MeanErrorTop7	$\frac{1}{7}\sum\{\text{Mass errors of the 7 highest intensity fragment ion peaks}\}$
sqMeanErrorTop7	$(\text{MeanErrorTop7})^2$
StdevErrorTop7	Standard deviation of mass errors of the 7 highest intensity fragment ion peaks
ChargeN	Boolean denoting whether peptide charge is N
enzN	Boolean denoting whether N-terminal agrees with cleavage rules
enzC	Boolean denoting whether C-terminal agrees with cleavage rules
enzInt	Number of internal cleavage sites

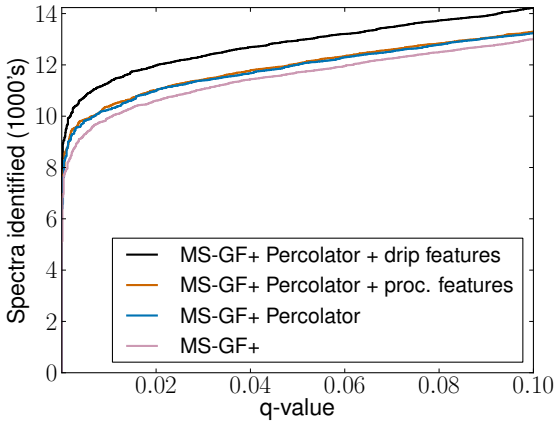


(a) Worm-1

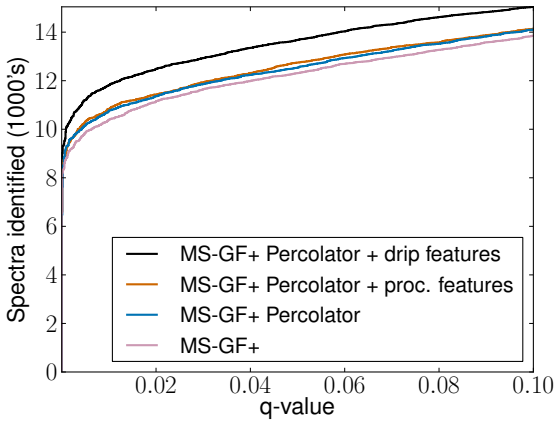
(b) Worm-2



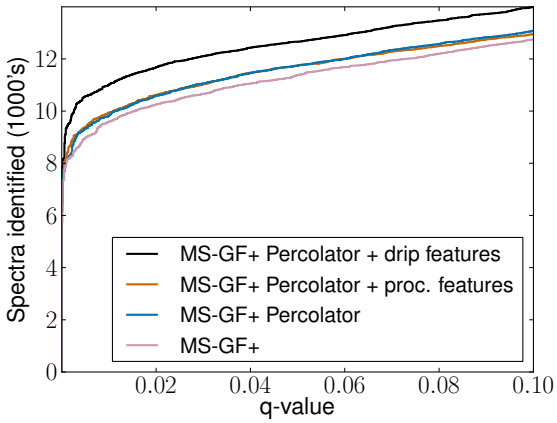
(c) Worm-3



(d) Yeast-1



(e) Yeast-2



(f) Yeast-3

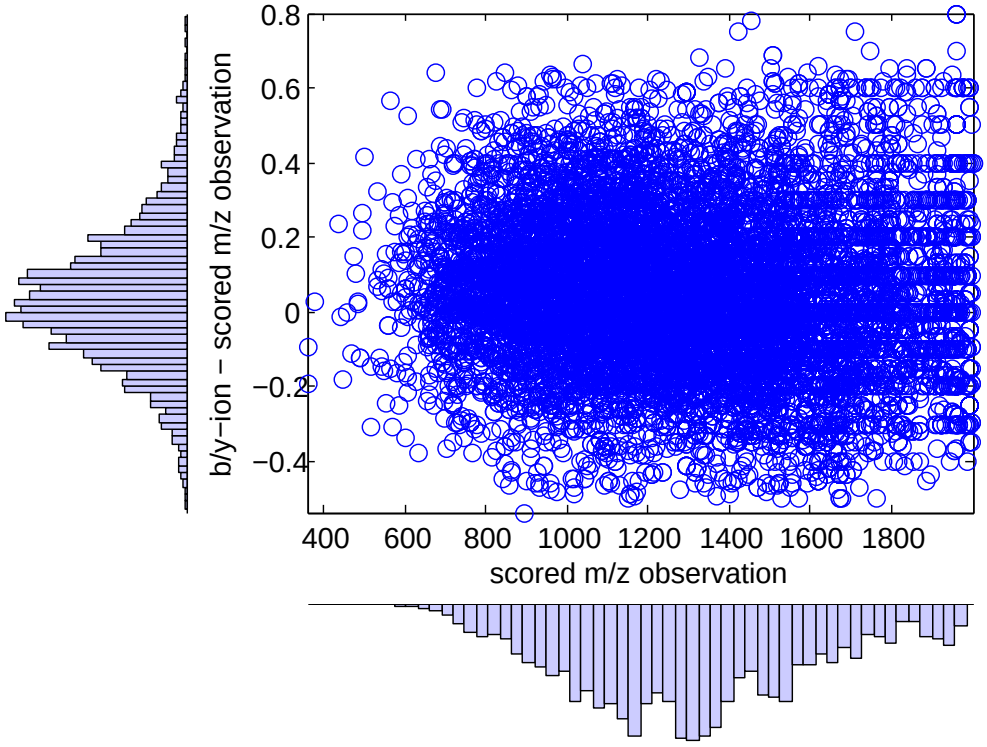
Figure 9.3: MS-GF+ utilizing the features listed in Table 9.3 as well DRIP derived features.

9.2 To bin or not to bin: soft versus hard decisions about theoretical peak locations

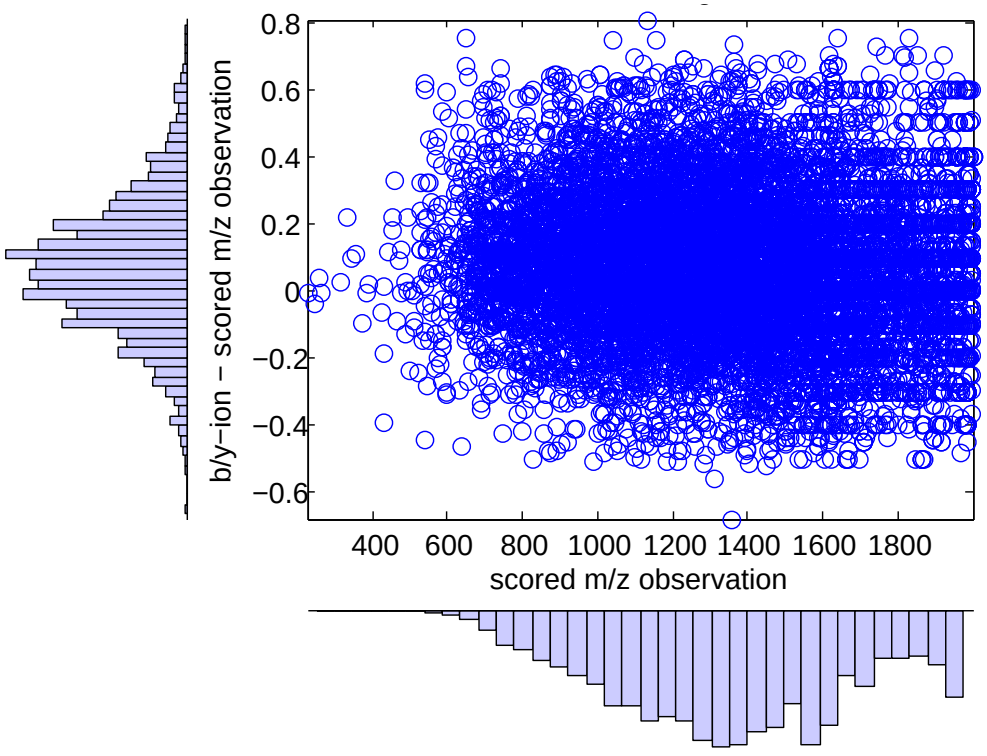
The vast majority of scoring algorithms quantize the observed spectrum or fixed-width threshold to align the theoretical and observed spectra. This means that outside of a pre-specified range (typically ~ 1 Th for low-resolution fragment ions), observed peaks may not be scored by theoretical peaks, i.e., may not be considered a *fragment ion match*, as it is often referred to in the literature. Such quantization and fixed-width thresholding schemes may be thought of as *hard decisions* regarding fragment ion matches; past a specified point, observed peaks may never be considered a match. The scoring functions which utilize such hard decisions may be thought of as generating a *static alignment* between the theoretical and observed spectra, such that there is only one allowable way in which theoretical peaks may score observed peaks. A much different approach is taken in DRIP.

In contrast, DRIP scores m/z values in their natural resolution, without discarding information due to quantization. Recall that, in DRIP, all possible alignments (i.e., sequences of insertions and deletions and the resulting scoring of observed peaks by theoretical peak Gaussians) are considered and the maximal such alignment is computed via the Viterbi algorithm. We may thus think of DRIP as dynamically aligning the theoretical and observed spectra, as we do not make any hard decisions regarding theoretical peak matches but rather infer the alignment which maximizes the log-likelihood. A theoretical peak Gaussian is free to score an observed peak very far in m/z, and it is the associated probability of such an event that matters. In this way, DRIP makes *soft decisions* with regards to theoretical peak matches, as all possible alignments between the theoretical and observed spectra are allowed and associated with a probability.

We've seen, in Figures 9.2 and 9.3, that using DRIP derived features leads to significantly better target/decoy classification accuracy than relying on quantization to derive such figures. We now show that soft decisions allow DRIP to consider fragment ion matches that hard decisions may not. Figure 9.4 displays the quantities used to derive the DRIP `sumScoredMz`



(a) Worm-1.



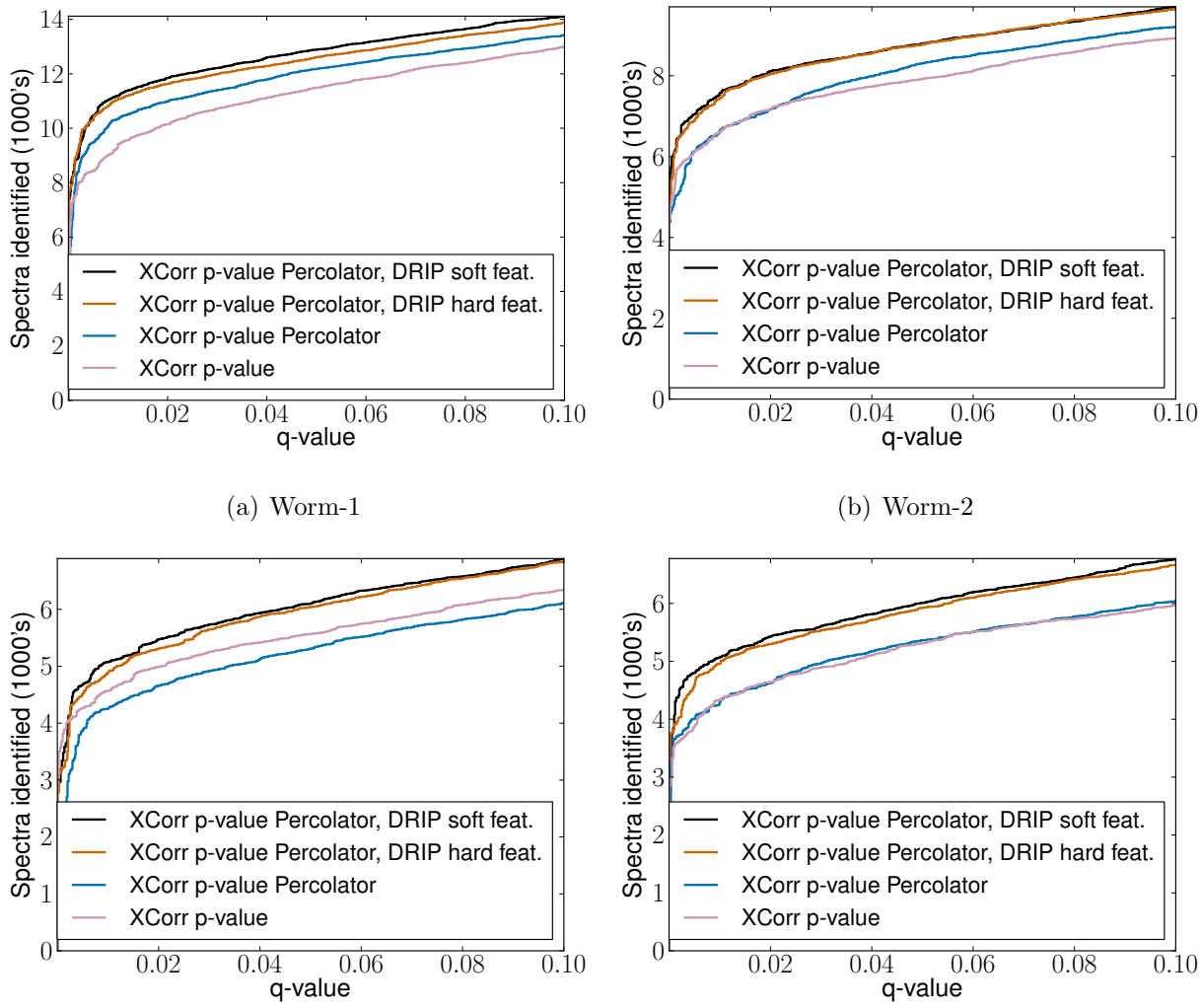
(b) Worm-2.

Figure 9.4: DRIP mean differences used to calculate feature `sumScoredMz`, for XCorr p-value PSMs.

for datasets Worm-1 and Worm-2. The x-axis for the scatter plots in Figure 9.4 corresponds to a scored m/z observation and the y-axis corresponds to the distance between a scored m/z observation and the mean of theoretical peak Gaussian used to score this m/z observation (recall that 99.99% of the Gaussian mass lies within a ~ 1 Th). Making hard decisions, we would not see any fragment ion matches with a distance greater than ~ 0.5 Th from their scored m/z observations. However, we see that using soft decisions results in DRIP inferring many matches which are greater than ~ 0.5 Th, although, as expected, a majority of matches are within 0.5 Th. Indeed, the static alignment any other scheme considers with regards to the theoretical and observed spectra is also considered in DRIP and weighted amongst all other possible alignments, so that the DRIP alignment strategy is strictly more general.

We show that matches inferred outside the bounds corresponding to hard decisions, as is possible with soft decisions, are beneficial for identification accuracy. Figure 9.5 displays the previously described DRIP features derived using soft decisions (denoted as "DRIP soft feat.") as well as DRIP features computed without including fragment ion matches further than 0.5 Th from the scoring theoretical peak Guassian mean (denoted as "DRIP hard feat."). While the two sets of features improve upon the standard set of XCorr p-value features, there is an appreciable loss in accuracy when we restrict the features to hard decisions.

Hard versus soft decisions have long been studied in machine learning and Bayesian statistics, particularly in the context of the EM algorithm. In standard EM, also called *soft EM*, probabilities are computed for each configuration of random variables and used during the algorithm. In stark contrast, *hard EM* algorithms assign random variables values with probability one. For instance, in the k-means algorithm, arguably the most popular hard EM algorithm, datapoints are assigned to clusters based on their distance from all k centroids, new centroids are computed based on cluster membership, and this process is repeated to convergence. Notions of hard and soft decisions are also generally defined for graphical models in terms of evidence [2], where observed children take their values with probability one or zero depending on the values of their parents (*hard evidence*) or observed children take their values with varying probabilities based on their parents (*soft evidence*, of which virtual evidence,



(a) Worm-1

(b) Worm-2

(c) Worm-3

(d) Worm-4

Figure 9.5: DRIP features derived using hard and soft decisions for XCorr p-values plus Percolator.

defined in Section 5.2, is a special case). This is the first work, to the authors' knowledge, that explores soft and hard decisions regarding MS/MS fragment ion matches, with DRIP being the first model to afford the ability to make soft decisions.

Chapter 10

CONCLUSIONS

In this work, we've shown that many of the current state-of-the-art MS/MS scoring functions may be described using GMs, from a GM representing any possible linear scoring function (Section 6.1) to a GM with which exact, nonparametric p-values for linear scoring functions may be computed (Section 6.1.1). The generality afforded by using GMs has allowed us to show that linear scoring functions may be significantly sped up by roughly an order of magnitude using trellises and beam pruning techniques (Section 7), both for low-resolution and high-resolution data. We seek a similar speed up using the GM for exact, nonparametric p-values in future work.

We have presented two GMs specifically designed to improve peptide identification of tandem mass spectra, Didea (Section 6.2) and DRIP (Section 6.3). For Didea, whose scoring function was designed to be a probabilistic analogue of XCorr, we derive upper and lower bounds regarding its conditional log-likelihood, including one which directly relates its scoring function to XCorr. Both Didea and DRIP allow effective parameter estimation (Section 8), in contrast to a majority of existing scoring functions which rely on hardcoded parameters. In the case of DRIP, generative training of model parameters significantly improves identification accuracy. DRIP's learning framework was also extended to the much more difficult but ultimately more accurate discriminative training. In the case of Didea, the conditional log-likelihood was reparameterized to be concave in the parameter to be learned, leading to significantly faster training runtime than the previous approach [63]. Remaining in the realm of concavity so that learning remains feasible, we showed that greatly expanding the set of parameters to be learned improved Didea identification accuracy. Due to both the generative representation of insertions and deletions and the use of max-product inference,

DRIP was also shown to provide valuable auxiliary information regarding PSMs, within each PSMs decoded Viterbi path.

Using the decoded sequences of insertions and deletions in DRIP, we've shown that we may extract features which may be used in tandem with a post-processor, such as Percolator, to significantly improved identification accuracy (Section 9.1). This makes DRIP particularly useful as a feature extractor for other methods and their standard set of post-processing features. Compared to simply calculating these features by quantizing the observed spectrum, much of the performance improvement was shown to stem from extracting these features by performing inference in DRIP, where m/z observations are not quantized. Furthermore, DRIP represents a general alignment strategy where we do not make hard decisions about where we expect fragment ions to lie, but rather penalize observations far from expected fragment ion locations (soft decisions). This leads to DRIP discovering fragment ion matches which are beyond the binning and threshold tolerances used by other methods and which are beneficial to the extracted features and post-processing accuracy (Section 9.2). This is the first exploration of such soft decisions for fragment ion matches within a scoring algorithm, and shows the utility of this approach.

10.1 Future work

In total, graphical models have been shown to be versatile tools capable of both speeding up and improving the accuracy of tandem mass spectra identification. The work described herein lays the foundation for the following avenues exploring future speed and accuracy improvements.

10.1.1 Denoising observed spectra

In practice, the number of observed peaks per spectrum is typically in the hundreds, so that the number of insertions per PSM is typically an order of magnitude larger than the number of fragment ion matches, leading to a low percentage of identified peaks (PIP), as displayed for XCorr p-values in Table 10.1. We note that for the yeast datasets, which contain

Table 10.1: XCorr p-value top PSM statistics (collected from the output of `tide-search`), including the percentage of identified peaks (PIP).

Dataset	Average b/y matches	Average number of observed peaks	Average PIP
Yeast-1	11.29	298.85	0.048
Yeast-2	11.22	301.63	0.047
Yeast-3	11.28	302.68	0.046
Yeast-4	11.26	303.8	0.047
Worm-1	12.07	772.96	0.018
Worm-2	12.65	794.26	0.0199
Worm-3	11.81	744.55	0.0197
Worm-4	11.88	707.57	0.02

relatively few observed peaks and thus far fewer insertions than the worm data, XCorr p-value is competitive with DRIP (Figure 6.8), but for the worm datasets, which contain far more noise peaks, XCorr p-value is significantly outperformed by DRIP. We conjecture that this discrepancy is due to the low PIP of XCorr p-value PSMs for the worm data and DRIP's ability to accurately estimate noise peaks in the observed spectrum (thus allowing DRIP to accurately identify peptides in low PIP spectra). As we've seen in Sections 9.1 and 9.2, DRIP is highly accurate at estimating insertions and deletions. However, DRIP's scoring function is not calibrated, leading us to depend on heuristics to calibrate PSM scores (Section 6.3.4). In order to combine the accurate information extracted in DRIP with principled calibration methods, we propose the following two stage system which first utilizes DRIP to remove noise in the observed spectrum then utilizes a calibrated scoring algorithm, such as XCorr p-values, to compute calibrated scores. This system also affords exploring how improving PIP impacts scoring function accuracy, addressing the earlier conjecture.

Given a peptide, DRIP will be utilized as a preprocessor of the observed spectrum,

removing a large number of insertions to increase the PIP of PSMs. In order to allow for some inaccuracy in the estimation of insertions, we will remove all but some small percentage of inserted observed peaks. Futhermore, some observed spectra noise is valuable information for identifying peptides, as seen in the improved accuracy utilizing only XCorr's foreground score versus including XCorr's background score. Critical to this work will be determining the percentage of insertions to be filtered per dataset and determining how insertions should be filtered (e.g., should we rank and filter insertion peaks by intensity or by distance from b/y-ion matches). Once denoised using DRIP, the observed spectrum will then be fed into a calibrated algorithm to produce calibrated scores, such as XCorr p-values, yielding calibrated scores identified at reliable PIPs.

This work has the potential to improve search accuracy relative to both search algorithms (DRIP and the calibrated scoring algorithm), as we are combining both the accuracy of DRIP's insertion estimation with principled calibration methods (i.e., exact nonparametric p-values). We note that DRIP may also be used to preprocess observed spectra for noncalibrated scoring functions (such as XCorr or X!Tandem) and such preprocessing may improve scoring function performance by improving each PSM's PIP (XCorr's PIP closely resemble those in Table 10.1), though it would be surprising if performance were better than a standard DRIP search since DRIP already significantly outperforms uncalibrated scoring algorithms.

10.1.2 Further extensions to Didea's learning framework

In Section 8.2, we reparameterized Didea's scoring function to be concave in the sole parameter to be learned. We then increased the number of learnable parameters, while retaining a concave objective function. Learning the expanded number of parameters lead to some improved search accuracy, but most importantly lead to a significant improvement in training time compared to the grid search utilized in [63]. In future work, we plan to further build on Didea's improved learning framework, exploring both L1 and L2 regularization, in which case the objective remains concave and may be feasibly trained. Regularization is a common technique in machine learning, where a regularizer is added to the objective function so

that solutions tend towards a particular form. In the case of L2 regularization, many of our learned weights will tend to be small in magnitude, and in the case of L1 regularization, many of our learned weights will tend to be zero. Recall that the expanded set of learned parameters in Didea correspond to weights for each shift of the observed spectrum. Note that for the L1 regularized shift weights, a learned zero weight means that particular shift is of little or no importance to the optimized scoring function. Thus, study of the L1 regularized parameters in Didea may also lead to insights to the related (Section 6.2.5) XCorr scoring function. Particularly, if it is the case that a large number of L1 regularized shift weights are zero and learning such parameters is beneficial for search accuracy, a similar framework where we learn a sparse vector of weights per shift for XCorr may also prove beneficial.

10.1.3 Deep DRIP

Fueled by the greatly improved computational resources of the past decade, particularly graphical processing units (GPUs), deep learning has re-emerged as a state-of-the-art technology, raising the performance bar for difficult tasks such as automatic speech recongition (ASR) [53] and image classification [50]. As deep neural networks (DNNs) have been utilized in ASR to replace the emission distributions in DBN speech decoders (what are called *hybrid DBN-DNN* models), a similar approach may be taken utilizing DRIP. Particularly, we propose replacing DRIP's conditional Gaussian distributions used to score observed peaks with DNNs.

Similar to ASR, given high-confidence training PSMs, DRIP would first be discriminatively trained, then the Viterbi path computed for each training PSM. The DRIP decoded sequence of fragment ions may then be used as labels and the observed peak observations as inputs to a DNN. In order to handle insertions, we may assign such peaks a unqiue class label (though labeling an observation with both the fragment ion and binary insertion value is also a possibility) to learn the insertion penalty. However, there is no guarantee the learned insertion penalty will provide the necessary tradeoff to accurately allow DRIP to distinguish between insertions and fragment ion matches (i.e., the network may learn that insertions should receive better scores than noninsertion observations), in which case deriving a uniform

penalty from the range of learned DNN scores is a possible solution.

This is perhaps the most ambitious proposed work, as it is nontrivial to effectively tune the many hyperparameters in a deep network (number of layers, number of units per layer, stochastic gradient descent learning schedule, number of training epochs, etc.) for new applications. The large state space of the labels (fragment ions) will require a deep and wide network, a very large amount of high-confidence training PSMs (this is critical), and significant training time. Furthermore, DRIP for high-resolution MS2 spectra (Section 6.3.6) does not have a fixed set of theoretical Gaussians (the set of Gaussian means is the set of unquantized b/y-ion values change for each candidate peptide set). While this may be rectified by fixing the set of high-resolution Gaussians via only retaining up to some pre-specified number of significant digits (for instance, retaining up to two significant figures equates to 200,000 theoretical Gaussians), this necessarily leads to a larger number of DNN class labels to be learned. This is a difficult hurdle, as DRIP's 2,000 low-resolution theoretical spectrum class labels is already large for a DNN to effectively learn in practice (for context, [53] utilized five hidden layers of 2,048 units to learn 183 class labels for ASR), so that training a high-resolution DNN will require either an even larger network and significantly more training data than the low-resolution case, or some other strategy to handle this high-dimensionality must be considered.

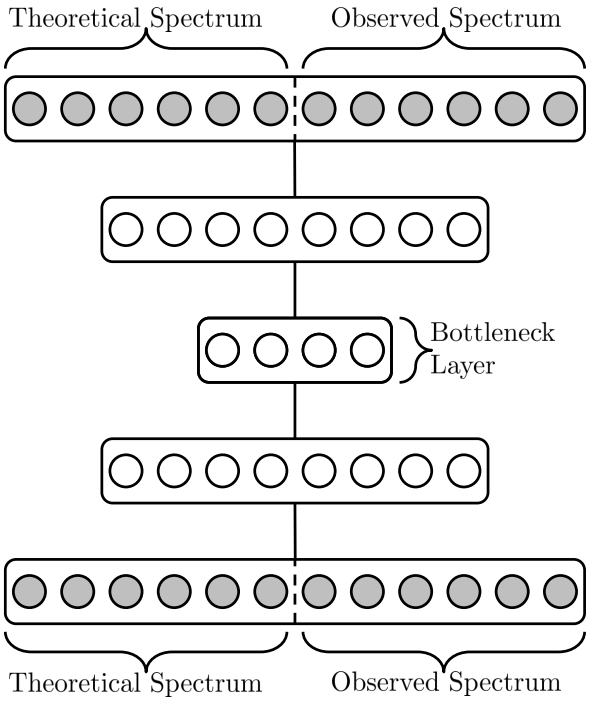
Finally, we note that the training data must provide sufficient evidence to learn each class label, such that each fragment ion match corresponding to a class label be represented in the training data. Otherwise, some alternate strategy must be proposed to learn those class labels for which the DNN does not have sufficient examples to effectively learn. In order to determine that the deep network is effectively learning parameters, one may check that the network assigns higher scores to observations close, in m/z value, to the class label (i.e., m/z observations score better the closer they are to fragment ions).

10.1.4 Deep feature extraction

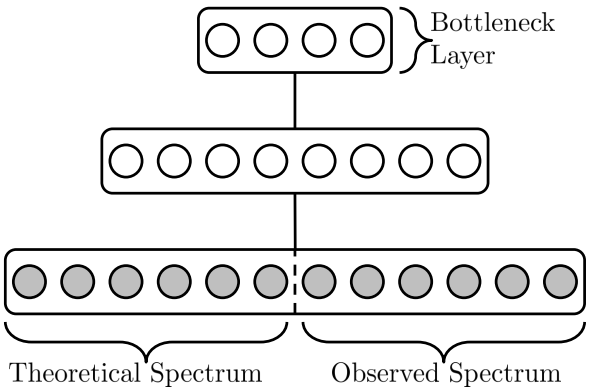
We've seen that DRIP may be utilized to extract accurate features describing PSMs. Such DRIP extracted features may be used in Percolator for significantly improved score recalibration. This begs the question whether we can extract other accurate features which may help Percolator better distinguish between the target and decoy classes during training. In order to further explore PSM feature extraction, we propose utilizing deep models, which have been used to great effect to extract features in other domains, such as ASR [54, 64, 62]. Central to these feature extractors are the training of deep autoencoders [30], deep networks which learn a lower-dimensional representation of the data of interest.

In a typical deep autoencoder, the input and output layers are tied. The middle layer of the deep autoencoder is called the *bottleneck* layer, and typically consists of significantly fewer units than the input/output layers. The layers from the input layer to the bottleneck layer typically consist of fewer and fewer units, which is similar for the layers going from the bottleneck layer to the output layer (the network is often symmetric about the bottleneck layer, and the top half of the network may even be tied with the bottom half, excluding the bottleneck, as a form of regularization). Thus, from the input to the bottleneck layer, the network is learning a lower and lower dimensional representation of the data, and from the bottleneck to the output layer, the network is learning how to reconstruct the data from the lower dimensional to the original dimensional space. The learned feature activations of the bottleneck layer may then be used to accurately encode the data in an incredibly compact form.

We propose using deep autoencoders to learn a lower dimensional representation of PSMs, and utilizing the compact representation as features to help classification in Percolator. The proposed deep autoencoder is depicted in Figure 10.2(a). The input and output layers consist of vector representations of the theoretical and observed spectra (in practice, the input and output dimension would thus be 4000). The bottleneck layer would be of a significantly smaller dimension than the input and output layers, though the bottleneck size in practice,



(a) PSM deep autoencoder.



(b) PSM deep feature extractor.

Figure 10.1: Deep autoencoder to learn mapping of PSMs to lower dimensional space and subsequent network for feature extraction. The actual number of layers and units per layer will vary, but the hourglass shape of the model will likely remain the same.

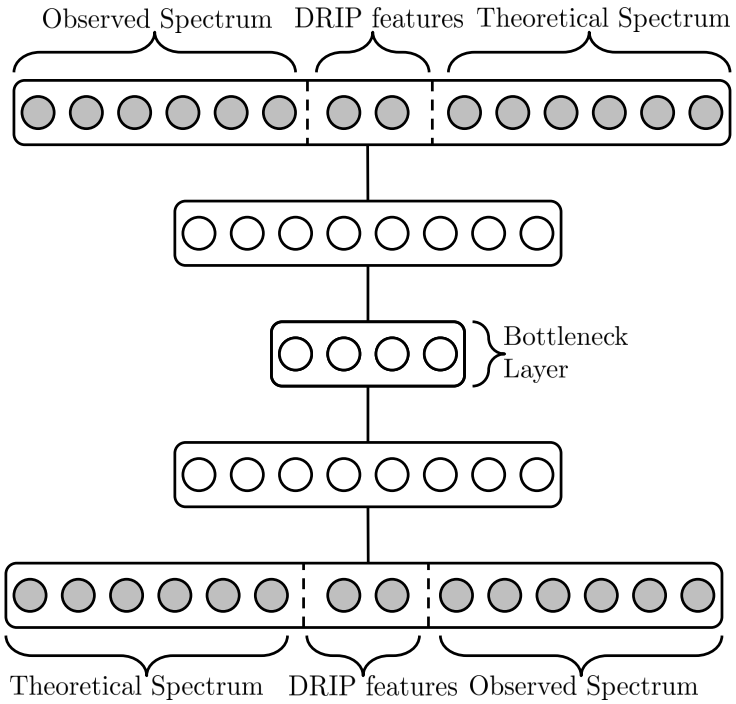
as well as the tapering envelope of all other hidden layers, require exploration. The feature extractor is depicted in Figure 10.2(b) where, once training of the autoencoder is completed, we remove the top half of the network and use the feature activations of the bottleneck layer as our extracted features. We may think of the network as condensing the information in the input layer, removing redundancy, and boiling the data down into a dense, compact feature vector.

The inclusion of accurate features detailing PSMs may further improve the quality of the feature representation learned by the autoencoder. To this end, we propose the model variant in Figure 10.2, where the input and output layers both include the highly accurate DRIP extracted PSM features. In this case, it will likely be desirable to exclude the DRIP extracted features from Percolator in order to avoid redundancy in the features used for classification (the deep network extracted features now encode the DRIP features). We note that this is not exclusive to DRIP extracted features, as any set of Percolator features may be used during autoencoding. In this way, we may use standard sets of Percolator features, such as those used for XCorr or MS-GF+ PSMs, to train the deep autoencoder. For such cases, it may be necessary to increase the bottleneck layer size, as well as solely use the learned feature representation in Percolator (to avoid redundancy, as previously mentioned).

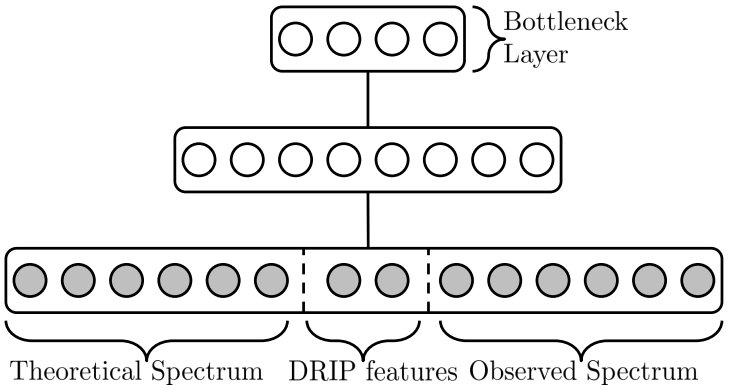
As noted in Section 10.1.3, tuning the many hyperparameters of the deep network is nontrivial and will take much work to explore and analyze. A very large number of high-confidence PSMs will also be necessary to train the network, as well as a large amount of compute time. However, once the network is trained, we may extract features using the models in Figures 10.1(b) and 10.2(b).

10.1.5 Faster p-value computation for linear scoring functions

Calibrating scores by computing exact, nonparametric p-values (Section 3.1.4) has significantly improved identification accuracy of the base linear scoring functions, perhaps best exemplified by comparing the accuracy XCorr versus the accuracy of XCorr p-value (Figures 4.1 and 4.2). However, this improvement in accuracy comes at a cost in runtime, as computing the



(a) PSM deep autoencoder, also using DRIP features.



(b) PSM deep feature extractor, also using DRIP features.

Figure 10.2: Deep autoencoder to learn mapping of PSMs to lower dimensional space and subsequent network for feature extraction, also utilizing DRIP features. Note that Perolcator features, such as those standardly used with XCorr and MS-GF+, may also be used. The actual number of layers and units per layer will vary, but the hourglass shape of the model will likely remain the same.

dynamic programming matrix for each observed spectrum requires significantly more time then scoring and ranking peptide candidates by XCorr (or by the linear scoring function to be calibrated) [33]. In this work, we've described how the scoring and ranking of peptide candidates by linear scoring functions may be significantly sped up by graphical models utilized in tandem with trellises and beam pruning, approximate inference techniques (Section 7). We now describe how graphical models and variational inference may be utilized to potentially speed up computation of the distribution of peptide scores used to compute p-values.

Recall the graphical model described in Section 6.1.1, which computes the distribution relative to a linear scoring function of all peptide scores of a particular precursor mass, given the observed spectrum. The distribution of interest (equivalent to the last column of the dynamic programming matrix) is the posterior of a random variable in the epilogue which corresponds to the total XCorr of a peptide accumulated in successive frames, computed via sum-product inference. For graphical models, variational methods [36, 70] are a family of approximate inference algorithms wherein we estimate the posterior distribution of interest via a surrogate distribution which is often much easier to compute. Key to this approach is the parameterization of the surrogate distribution via variational parameters and casting the learning of these parameters as an optimization problem. This optimization problem (maximization of the extended lower bound, or ELBO) is equivalent to minimizing, with respect to the variation parameters, the Kullback-Leibler (KL) divergence between the posterior of interest and the surrogate distribution, thus ensuring these two distributions are "close" (as measured by the KL divergence).

Variational inference has been used to great success, often in cases where the posterior of interest is intractable to compute but utilizing the surrogate function is feasible [10, 18, 31], and its utiliziation would be novel to MS/MS. Furthermore, this would serve as a critical speedup to MS/MS analysis, as nonparametric p-values for linear scoring functions has been a topic of great interest in the field but are much slower, in practice, to compute than the uncalibrated linear scores of all peptide candidates [33]. Assuming a significant speed up using variational methods to compute the posterior necessary to calculate nonparametric

p-values, graphical models would thus significantly improve search times for the scoring and ranking of peptides (Section 7) as well as the principled calibration of these PSMs via p-value computation.

10.1.6 Approximate Didea p-values

Recall the linear upper bound to Didea's scoring function, derived in Section 6.2.5. Further study of this upper bound, for which we may efficiently compute exact p-values of, may allow us to approximate p-values for the original, non-linear Didea scoring function. We describe a variational approach, wherein we approximate the distribution of interest (which is computationally difficult to compute in practice) via a simpler surrogate function which is much more tractable. Critical to this strategy is that the surrogate function upper (or lower) bound the distribution of interest, so that minimizing (or maximizing) the surrogate function with respect to variational parameters leads to an accurate estimate of the distribution of interest. Bounds on Didea's scoring function provide future avenues to approximate p-values for Didea's complicated scoring function. Thus, in future work, we plan to parameterize Didea's upper bound (which is linear) and utilize the minimized (with respect to variational parameters) upper bound as a surrogate for Didea, efficiently computing p-values for the surrogate function which serve as approximate p-values for Didea.

BIBLIOGRAPHY

[1] Galen Andrew and Jeff Bilmes. Memory-efficient inference in dynamic graphical models using multiple cores. In *Fifteenth International Conference on Artificial Intelligence and Statistics (AISTAT)*, La Palma, Canary Islands, April 2012.

[2] J. Bilmes. On soft evidence in Bayesian networks. Technical report, Department of Electrical Engineering, University of Washington, 2004. UWEETR-2004-0016.

[3] J. Bilmes and G. Zweig. The Graphical Models Toolkit: An open source software system for speech and time-series processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002.

[4] Jeff Bilmes. Dynamic Bayesian multinets. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 38–45, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[5] Jeff Bilmes. Dynamic graphical models. *IEEE Signal Processing Magazine*, 27(6):29–42, Nov 2010.

[6] Jeff Bilmes and Chris Bartels. Graphical model architectures for speech recognition. *IEEE Signal Processing Magazine*, 22(5):89–100, September 2005.

[7] Jeff A. Bilmes. *Dynamic Graphical Models and the Graphical Models Toolkit (GMTK)*. University of Washington, Seattle, 2014.

[8] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[9] John Binder, Kevin Murphy, and Stuart Russell. Space-efficient inference in dynamic probabilistic networks. *Bclr*, 1:t1, 1997.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(5):993–1022, 2003.

[11] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[12] M. Brosch, L. Yu, T. Hubbard, and J. Choudhary. Accurate and sensitive peptide identification with Mascot Percolator. *Journal of Proteome Research*, 2009. In press.

[13] H. Choi and A. I. Nesvizhskii. Semisupervised model-based validation of peptide identifications in mass spectrometry-based proteomics. *Journal of Proteome Research*, 7(1):254–265, 2008.

[14] R. Craig and R. C. Beavis. Tandem: matching proteins with tandem mass spectra. *Bioinformatics*, 20:1466–1467, 2004.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–22, 1977.

[16] B. Diament and W. S. Noble. Faster sequest searching for peptide identification from tandem mass spectra. *Journal of Proteome Research*, 10(9):3871–3879, 2011. PMC3166376.

[17] Y. Ding, H. Choi, and A. Nesvizhskii. Adaptive discriminant function analysis and reranking of MS/MS database search results for improved peptide identification in shotgun proteomics. *Journal of Proteome Research*, 7(11):4878–4889, 2008.

[18] Finale Doshi, Kurt Miller, Jurgen V Gael, and Yee W Teh. Variational inference for the indian buffet process. In *International Conference on Artificial Intelligence and Statistics*, pages 137–144, 2009.

[19] Jarrett D Egertson, Jimmy K Eng, Michael S Bereman, Edward J Hsieh, Gennifer E Merrihew, and Michael J MacCoss. De novo correction of mass measurement error in low resolution tandem ms spectra for shotgun proteomics. *Journal of The American Society for Mass Spectrometry*, pages 1–8, 2012.

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

[21] Jimmy K Eng, Tahmina A Jahan, and Michael R Hoopmann. Comet: An open-source ms/ms sequence database search tool. *Proteomics*, 13(1):22–24, 2013.

[22] Joseph Gallian. *Contemporary abstract algebra*. CengageBrain. com, 2010.

[23] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. Xu, D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant. Open mass spectrometry search algorithm. *Journal of Proteome Research*, 3:958–964, 2004. OMSSA.

[24] Lewis Y. Geer, Sanford P. Markey, Jeffrey A. Kowalak, Lukas Wagner, Ming Xu, Dawn M. Maynard, Xiaoyu Yang, Wenyao Shi, and Stephen H. Bryant. Open mass spectrometry search algorithm. *Journal of Proteome Research*, 3(5):958–964, 2004.

[25] V. Granholm, W. S. Noble, and L. Käll. A cross-validation scheme for machine learning algorithms in shotgun proteomics. *BMC Bioinformatics*, 13(Suppl 16):S3, 2012. PMC3489528.

[26] Viktor Granholm, Sangtae Kim, Jose CF Navarro, Erik Sjolund, Richard D Smith, and Lukas Kall. Fast and accurate database searches with ms-gf+ percolator. *Journal of proteome research*, 13(2):890–897, 2013.

[27] John T. Halloran. *Training HMMs in GMTK*. University of Washington, Seattle, 2015.

[28] John T. Halloran, Jeff A. Bilmes, and William S. Noble. Learning peptide-spectrum alignment models for tandem mass spectrometry. In *Uncertainty in Artificial Intelligence (UAI)*, Quebic City, Quebec Canada, July 2014. AUAI.

[29] Md Rafiul Hassan and Baikunth Nath. Stock market forecasting using hidden markov model: a new approach. In *Intelligent Systems Design and Applications, 2005. ISDA'05. Proceedings. 5th International Conference on*, pages 192–196. IEEE, 2005.

[30] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[31] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013.

[32] Michael M Hoffman, Orion J Buske, Jie Wang, Zhiping Weng, Jeff A Bilmes, and William Stafford Noble. Unsupervised pattern discovery in human chromatin structure through genomic segmentation. *Nature Methods*, 9:473–476, 2012.

[33] J Jeffry Howbert and William S Noble. Computing exact p-values for a cross-correlation shotgun proteomics score function. *Molecular & Cellular Proteomics*, pages mcp–O113, 2014.

[34] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2001.

[35] Gang Ji, Jeff Bilmes, Jeff Michels, Katrin Kirchhoff, and Chris Manning. Graphical model representations of word lattices ieee/acl 2006 workshop on spoken language technology (slt2006) , palm beach, aruba, dec 2006. *IEEE/ACL Workshop on Spoken Language Technology (SLT)*, 2006.

[36] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. Introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

[37] Michael I Jordan. Graphical models. *Statistical Science*, pages 140–155, 2004.

[38] L. Käll, J. Canterbury, J. Weston, W. S. Noble, and M. J. MacCoss. A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets. *Nature Methods*, 4:923–25, 2007.

[39] L. Käll, J. D. Storey, M. J. MacCoss, and W. S. Noble. Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. *Journal of Proteome Research*, 7(1):29–34, 2008.

[40] Uri Keich, Attila Kertesz-Farkas, and William Stafford Noble. Improved false discovery rate estimation procedure for shotgun proteomics. *Journal of proteome research*, 14(8):3148–3161, 2015.

[41] Uri Keich and William Stafford Noble. On the importance of well-calibrated scores for identifying shotgun proteomics spectra. *Journal of proteome research*, 14(2):1147–1160, 2014.

[42] A. Keller, A. I. Nesvizhskii, E. Kolker, and R. Aebersold. Empirical statistical model to estimate the accuracy of peptide identification made by MS/MS and database search. *Analytical Chemistry*, 74:5383–5392, 2002.

[43] Sangtae Kim, Nikolai Mischerikow, Nuno Bandeira, J. Daniel Navarro, Louis Wich, Shabaz Mohammed, Albert J. R. Heck, and Pavel A. Pevzner. The generating function of cid, etd, and cid/etd pairs of tandem mass spectra: Applications to database search. *Molecular and Cellular Proteomics*, 9(12):2840–2852, 2010.

[44] Sangtae Kim and Pavel A Pevzner. Ms-gf+ makes progress towards a universal database search tool for proteomics. *Nature communications*, 5, 2014.

[45] M. Kinter and N. E. Sherman. *Protein sequencing and identification using tandem mass spectrometry*. Wiley-Interscience, 2000.

[46] A. A. Klammer, C. Y. Park, and W. S. Noble. Statistical calibration of the SEQUEST XCorr function. *Journal of Proteome Research*, 8(4):2106–2113, 2009. PMC2807930.

[47] A. A. Klammer, S. R. Reynolds, M. Hoopmann, M. J. MacCoss, J. Bilmes, and W. S. Noble. Modeling peptide fragmentation with dynamic Bayesian networks yields improved tandem mass spectrum identification. *Bioinformatics*, 24(13):i348–i356, 2008.

[48] D. Koller, N. Friedman, L. Getoor, and B. Taskar. Graphical models in a nutshell. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[49] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus among words: lattice-based word error minimization. In *Eurospeech*, 1999.

[52] Sean McIlwain, Kaipo Tamura, Attila Kertesz-Farkas, Charles E Grant, Benjamin Diament, Barbara Frewen, J Jeffry Howbert, Michael R Hoopmann, Lukas Käll, Jimmy K Eng, et al. Crux: rapid open source protein tandem mass spectrometry analysis. *Journal of proteome research*, 2014.

[53] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14–22, 2012.

[54] Nelson Morgan. Deep and wide: Multiple layers in automatic speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):7–13, 2012.

[55] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

[56] H. Ney and S. Ortmanns. Progress in dynamic programming search for lvcsr. *Proceedings of the IEEE*, 88(8):1224–1240, 2000.

[57] L. Pauling, H. A. Itano, S. J. Singer, and I. C. Wells. Sickle Cell Anemia, a Molecular Disease. *Science*, 110:543–548, November 1949.

[58] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[59] Daniel Povey. *Discriminative training for large vocabulary speech recognition*. PhD thesis, University of Cambridge, 2003.

[60] B. Y. Renard, M. Kirchner, F. Monigatti, A. R. Ivanov, J. Rappsilber, D. Winterc, J. A.J. Steen, F. A. Hamprecht, and H. Steen. When less can yield more–computational preprocessing of ms/ms spectra for peptide identification. *Proteomics*, 9(21):4978–4984, 2009.

[61] Richard Rogers, Jeff A. Bilmes, and John T. Halloran. *GMTK Tutorial on Dynamic Graphical Model Training with Gaussian Mixture Unaries, using TIMIT*. University of Washington, Seattle, 2015.

[62] Tara N Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Auto-encoder bottleneck features using deep belief networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4153–4156. IEEE, 2012.

[63] Ajit P. Singh, John Halloran, Jeff A. Bilmes, Katrin Kirchoff, and William S. Noble. Spectrum identification using a dynamic bayesian network model of tandem mass spectra. In *Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, USA, July 2012. AUAI.

[64] Garimella SVS Sivaram and Hynek Hermansky. Sparse multilayer perceptron for phoneme recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):23–29, 2012.

[65] M. Spivak and W. S. Noble. Learning score function parameters for improved spectrum identification in tandem mass spectrometry experiments. *Journal of Proteome Research*, 11(9):4499–4508, 2012. PMC3436966.

[66] M. Spivak, J. Weston, L. Bottou, L. Käll, and W. S. Noble. Improvements to the Percolator algorithm for peptide identification from shotgun proteomics data sets. *Journal of Proteome Research*, 8(7):3737–3745, 2009. PMC2710313.

[67] M. Spivak, J. Weston, D. Tomazela, M. J. MacCoss, and W. S. Noble. Direct maximization of protein identifications from tandem mass spectra. *Molecular and Cellular Proteomics*, 11(2):M111.012161, 2012. PMC3277760.

[68] H. Steen and M. Mann. The ABC's (and XYZ's) of peptide sequencing. *Nature Reviews Molecular Cell Biology*, 5:699–711, 2004.

[69] Andreas Stolcke, Yocahi Konig, and Mitchel Weintraub. Explicit word error minimization in n-best list rescoring. In *Eurospeech*, volume 97, pages 163–166, 1997.

[70] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. Technical Report 649, UC Berkeley, Dept. of Statistics, 2003.

[71] Mark C Walters, Melinda Patience, Wendy Leisenring, James R Eckman, J Paul Scott, William C Mentzer, Sally C Davies, Kwaku Ohene-Frempong, Françoise Bernaudin, Dana C Matthews, et al. Bone marrow transplantation for sickle cell disease. *New England Journal of Medicine*, 335(6):369–376, 1996.

[72] S. Wang, J. T. Halloran, J. A. Bilmes, and W. S. Noble. Faster graphical model identification of tandem mass spectra using peptide word lattices. 2014.

[73] C. D. Wenger and J. J. Coon. A proteomics search algorithm specifically designed for high-resolution tandem mass spectra. *Journal of proteome research*, 2013.

[74] Linfeng Wu, Sophie I Candille, Yoonha Choi, Dan Xie, Lihua Jiang, Jennifer Li-Pook-Than, Hua Tang, and Michael Snyder. Variation and genetic control of protein abundance in humans. *Nature*, 499(7456):79–82, 2013.

[75] Mingguo Xu, Zhendong Li, and Liang Li. Combining percolator with x! tandem for accurate and sensitive peptide identification. *Journal of proteome research*, 12(6):3026–3033, 2013.

Appendix A

INFERENCE IN MS/MS GRAPHICAL MODELS: GMTK COMMAND LINES

We provide the Graphical Models ToolKit (GMTK) command lines for the many different applications of GM inference for MS/MS peptide identification detailed in this work. For further discussion of defining GMs in GMTK and the many inference options supported by GMTK, please see [7]. After each GM is defined in what is called a structure file (denoted below with the extension `.str`), each model is triangulated (discussed in Section 5.1.2) using the executable `gmtkTriangulate` to produce a triangulate file (denoted below with the extension `.trifile`).

A.1 Sum-product inference in Didea: `gmtkJT`

Sum-product inference via the junction tree algorithm (detailed in Section 5.1.2) is used to calculate the posterior probability with which Didea utilizes to score peptides. The junction tree algorithm is supported in GMTK via the executable `gmtkJT`. Given the GM defined in Figure 6.3 and the triangulated model, a clique containing the single hidden variable τ, corresponding to the shift of the observed spectrum, is defined as the first clique in the epilogue of the triangulation file. The GMTK command line to calculate the posterior distribution of τ is thus

```
gmtkJT -strFile didea.str \
  -trifile didea.str.trifile \
  -inputMasterFile didea.mtr \
  -of1 quantizedObservedSpectrum -ni1 0 -nf1 2000 \
  -of2 peptideSequence -ni2 1 -nf2 0 \
  -fdiffact1 rl \
  -doDistributeEvidence T \
  -eCliquePrintRange 0
```

The arguments to `gmtkJT` are:

-strFile didea.str specifies the graph of the model

-trifile didea.str.trifile specifies the triangulated graph where, for Didea, a clique containing the single hidden variable τ, corresponding to the shift of the observed spectrum, is defined as the first clique in the epilogue

-inputMasterFile didea.mtr specifies the master file, which more specifically defines the relationship between random variables and their parents (see [7] for further details).

-of1 quantizedObservedSpectrum specifies the first observation file, whose index is denoted by the `1` proceeding `of` containing the observed spectrum observations which, for Didea, corresponds to a vector of quantized observations

-ni1 0 specifies the number of integers in the first observation file (none, since the quantized observed spectrum is a vector of real numbers)

-nf1 2000 specifies the number of floating point numbers in the first observation file (2000, representing 2000 bins along the m/z axis)

-of2 peptideSequence specifies the second observation file, containing the sequence of amino acids

-ni2 1 -nf2 0 specifies Didea has only one peptide observation per frame, an integer denoting an amino acid in the peptide sequence

-fdiffact1 rl specifies that the frames in Didea are defined in terms of the peptide sequence, so that the observed spectrum observations are made available in each frame

-doDistributeEvidence T specifies that a backward pass take place during sum-product inference (detailed in Section 5.1.2)

-eCliquePrintRange 0 specifies that the posterior of the first clique in the epilogue (defined in didea.str.trifile) be output

A.2 Viterbi inference in DRIP: `gmtkViterbi`

In DRIP (the graph of which is defined in Figure 6.5), max-product inference via the Viterbi algorithm is used to calculate the Viterbi path and Viterbi score of a peptide given the observed spectrum. The Viterbi algorithm is supported in GMTK via the executable `gmtkViterbi`. The GMTK command to calculate the Viterbi path of a PSM in DRIP is thus

```
gmtkViterbi -strFile drip.str \
  -trifile drip.str.trifile \
  -inputMasterFile drip.mtr \
  -vitValsFile psmViterbiPath \
  -of1 observedSpectrumPeaks -ni1 0 -nf1 2 \
  -of2 peptideLength -ni2 1 -nf2 0 \
  -inputTrainableParameters dripTrainedParameters
```

Many of the arguments are similar to those for Didea. Note that the vector of theoretical peaks is defined in `drip.mtr`. The new arguments to `gmtkViterbi` are:

-vitValsFile psmViterbiPath specifies to write the Viterbi path of each PSM in output file `psmViterbiPath`

-of1 observedSpectrumPeaks -ni1 0 -nf1 2 specifies that each frame in DRIP contains two floating point observations which correspond to the m/z value and intensity of an observed peak

-of2 peptideLength -ni2 1 -nf2 0 specifies that in each frame, we have an observed integer denoting the length of the theoretical spectrum

-inputTrainableParameters dripTrainedParameters specifies the learned theoretical means and variances utilized by DRIP, further detailed in Sections A.4 and A.5

A.3 Faster database search via approximate Viterbi inference in XCorr trellises: `gmtkViterbi` *with k-beam pruning*

GMTK supports specifying trellises to define the state of space of sequences of random variables (specified in the structure file, detailed in [7]). GMTK also supports many different types of beam pruning options for approximate inference, of which we use k-beam pruning (detailed in Section 7.0.7). Utilizing the GM defined in Figure 6.1, where the observed spectrum is quantized and preprocessed as in XCorr (described in Section 6.3.3), scoring sets of candidate peptides was significantly sped up, as described in Section 7, using max-product inference and beam pruning. The GMTK command line to calculate the most probable candidate peptide using k-beam pruning is thus

```
gmtkViterbi -strFile xcorrTrellis.str \
  -trifile xcorrTrellis.str.trifile \
  -inputMasterFile xcorrTrellis.mtr \
  -of1 quantizedObservedSpectrum -ni1 0 -nf1 2000 \
```

```
  -of2 peptideTheoreticalPeaks -ni2 1 -nf2 0 \
  -fdiffact1 rl \
  -ckbeam 10000
```

The arguments to `gmtkViterbi` (other than the structure, triangulation, and master file) are

-of1 quantizedObservedSpectrum -ni1 0 -nf1 2000 specifies, as in Didea, the vector of quantized and preprocessed observed spectrum identifications

-of2 peptideTheoreticalPeaks -ni2 1 -nf2 0 specifies a theoretical peak in each frame

-fdiffact1 rl specifies that the sequence of frames is defined in terms of the sequence of theoretical peaks, so that the observed spectrum observations are made available in each frame

-ckbeam 10000 specifies using k-beam pruning for appromximate inference, retaining only the top $k = 10000$ hypotheses in each frame

A.4 Generative training in DRIP: `gmtkEMtrain`

GMTK supports generative training of GMs via the expectation-maximization (EM) algorithm [15, 8] through the executable `gmtkEMtrain`. Continuing from the discussion in the earlier Appendix A.2, the GMTK command to train the Gaussian means and variances in DRIP is thus

```
gmtkEMtrain -strFile drip.str \
  -trifile drip.str.trifile \
  -inputMasterFile drip.mtr \
  -vitValsFile psmViterbiPath \
  -of1 observedSpectrumPeaks -ni1 0 -nf1 2 \
  -of2 peptideLength -ni2 1 -nf2 0 \
  -inputTrainableParameters dripHandTunedParameters \
  -outputTrainableParameters dripTrainedParameters \
  -maxEmIters 100 \
  -lldp 1.0e-6
```

Many of the arguments are the same from as those passed to `gmtkViterbi` in Appendix A.2, except for:

-inputTrainableParameters dripHandTunedParameters specifies that we initialize our parameters to those of the hand-tuned parameters discussed in Section 8.1.1

-outputTrainableParameters dripGeneratively TrainedParameters specifies where to write the output EM learned parameters

-maxEmIters 100 specifies that we run EM for a maximum of 100 iterations

-lldp 1.0e-6 specifies to stop EM if the log-likelihood difference (lldp) between successive iterations is less than 1.0e-6

A.5 Discriminative training in DRIP: `gmtkMMItrain`

GMTK supports Maximum Mutual Information Information (defined in Section 8.1.2) via the executable `gmtkMMItrain`, the command line of which is

```
gmtkMMItrain \
  -strFile drip.str \
  -triFile drip.str.trifile \
  -inputMasterFile drip.mtr \
  -strFile2 dripTrellis.str \
  -triFile2 dripTrellis.str.trifile \
  -inputMasterFile2 drip.mtr \
  -inputTrainableParameters dripGeneratively TrainedParameters \
  -inputTrainableParameters2 dripGeneratively TrainedParameters \
  -covarLr 1.0e-5 \
  -meanLr 1.0e-5 \
  -decayCovarLrRate 0.25 \
  -decayMeanLrRate 0.5
```

As detailed in Section 8.1.2, the numerator model is the same as that trained during generative training. So, as in Appendix A.4, the numerator model consists of specifying `drip.str`, `drip.str.trifile`, and `drip.mtr` as the structure file, triangulation file, and master file, respectively. The denominator model is defined to be, given the training database and

observed spectrum, the set of peptide candidates other than the high-confidence PSM, represented using a trellis. The denominator is thus specified as `dripTrellis.str` and `dripTrellis.str.trifile` for the structure file and triangulation file, respectively (the relationships between children and parents do not change, so the master file does not change). The remaining options regarding `gmtkMMItrain`, which specify the learning rate in stochastic gradient ascent, are discussed extensively in [61].

Appendix B

UPPER AND LOWER BOUNDS ON DIDEA'S SCORING FUNCTION

B.1 Second upper bound

Recall the discussion in Section 6.2.5 and please refer to Section 6.2 for general notation. Interestingly, we may compute an alternative upper bound to Equation 6.7 for Didea's scoring function. Let $h(\tau_0) = \mathrm{p}(x, z_\tau|\tau_0 = \tau)$. By Markov's inequality, we have

$$\begin{aligned}
\mathrm{Didea}(s,x) &= \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \log \frac{\alpha}{\alpha}\mathbb{E}_\tau[h(\tau_0)] \\
&\leq \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \log \alpha \mathrm{p}(h(\tau_0)\geq\alpha) \\
&\leq \log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \log \mathrm{p}(h(\tau_0)\geq\alpha) - \log\alpha,
\end{aligned}$$

where $\alpha > 0$. The distribution $\mathrm{p}(h(\tau_0))$ may be computed naively in $\mathcal{O}(|\mathcal{T}|n)$ time or much faster using more the algorithms in Appendix C), so that we may efficiently achieve the upper bound

$$\mathrm{Didea}(s,x) \leq \inf_\alpha \big(\log \mathrm{p}(x,z|\tau_0=0)\mathrm{p}(\tau_0=0) - \log \mathrm{p}(h(\tau_0)\geq\alpha) - \log\alpha \big). \tag{B.1}$$

The relationship between the upperbound in Equation B.1 and the XCorr upper bound in Equation 6.7 is currently unknown. These upper bounds may prove useful in future work focusing on computing p-values for Didea. The current computational difficulty is that Didea's scoring function is non-linear, and thus does not lend itself to efficient dynamic programming schemes to calculate p-values, such as those used employed by employed by MS-GF+ and XCorr p-value. By relaxing Didea's scoring function to a closely related, simpler function we may be able to bound p-values for Didea's original scoring function.

B.2 Lower bound

It is often useful if one can derive lower bounds to accompany upper bounds regarding a quantity. We now compute a lower bound for Didea's scoring function, which accompnanies the upper bounds in Sections 6.2.5 and B.1. Let $k = \sup_{\tau_0} h(\tau_0) + \epsilon$, for $\epsilon > 0$, and $d < \mathbb{E}[h(\tau_0)]$. Furthermore, assume $\mathbb{E}_\tau[h(\tau_0)] > 0$. By Markov's reverse inequality, we have $\frac{1}{k-d}(\mathbb{E}_\tau[h(\tau_0)] - d) \leq \mathrm{p}(\mathbb{E}_\tau[h(\tau_0)])$, so that

$$\begin{aligned}
\mathrm{Didea}(s,x) &= \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - \log \mathbb{E}_\tau[h(\tau_0)] \\
&= \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - \log \frac{k-d}{k-d}(\mathbb{E}_\tau[h(\tau_0)] - d) + \log\,(1 - \frac{d}{\mathbb{E}_\tau[h(\tau_0)]}) \\
&\geq \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - \log\,(k-d)\mathrm{p}(h(\tau_0) > d) + \log\,(1 - \frac{d}{\mathbb{E}_\tau[h(\tau_0)]}) \\
&\geq \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - \log\,(k-d)\mathrm{p}(h(\tau_0) > d) - \log \mathbb{E}_\tau[h(\tau_0)] \\
&\geq \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - \log\,(k-d)\mathrm{p}(h(\tau_0) > d) - \mathbb{E}_\tau[\log h(\tau_0)] \qquad \text{(B.2)} \\
&\geq \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - \log\,(k-d)\mathrm{p}(h(\tau_0) > d) - \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}}(b+y)^T z'_\tau \\
&\geq \log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) - (\log\,(k-d)\mathrm{p}(h(\tau_0) > d) + \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}}(b+y)^T z'_\tau),
\end{aligned}$$

where Equation B.2 is due to Jensen's inequality. $\mathrm{p}(h(\tau_0))$ may be computed in at most $\mathcal{O}(|\mathcal{T}|n)$ time and, as previously noted, may be more quickly computed using one of the algorithms in Appendix C, while $\sum_{\tau\in\mathcal{T}}(b+y)^T z'_\tau$ may be computed in $\mathcal{O}(n)$ time by caching $\sum_{\tau\in\mathcal{T}} z'_\tau$ (as in XCorr). We may thus compute the following in at most $\mathcal{O}(|\mathcal{T}|n)$ time,

$$\begin{aligned}
\mathrm{Didea}(s,x) \geq \sup_{d:d<\mathbb{E}[(h(\tau_0)]} \Big[&\log \mathrm{p}(x,z|\tau_0 = 0)\mathrm{p}(\tau_0 = 0) \\
&- \Big(\log\,(k-d)\mathrm{p}(h(\tau_0) > d) + \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}}(b+y)^T z'_\tau\Big)\Big]. \qquad \text{(B.3)}
\end{aligned}$$

Interestingly, the lower bound in Equation B.3 is a foreground score only involving the non-shifted version of the observed spectrum minus a background score involving an average over shifted versions of the observed spectrum, once more yielding a scoring function similar to XCorr.

Appendix C

EFFICIENT ALGORITHMS FOR CALCULATING ALL INVIDIDUAL DOT-PRODUCTS IN A SLIDING DOT-PRODUCT

Assume we have a collection of theoretical spectra, X, and an observed spectrum, $y \in \mathcal{R}^n$, where for $x \in X$, $x \in \mathcal{R}^n$. We would like to compute sliding dot-products (cross-correlations) for each $x \in X$. In the case of XCorr, we are interested in computing

$$\text{XCorr}(x, y) = x^T y - \frac{1}{75} \sum_{\tau=-75}^{75} x^T y_\tau = x^T (z - \frac{1}{75} \sum_{\tau=-75}^{75} y_\tau) = x^T y',$$

where the linearity of XCorr with respect to a theoretical spectrum allows the caching of y' so that computing $\text{XCorr}(x, y)$ $\forall x \in X$ is easily accomplished in $\mathcal{O}(|X|n)$ time. However, assume the scoring function we are interested in is not linear with respect to a theoretical spectrum. For instance, let $m = [x^T y_{-75}\ x^T y_{-74}\ \ldots\ x^T y_{75}]^T$ and m' be the vector containing the elements m in ascending sorted order, i.e., $m'(0) \geq m'(1) \geq \cdots \geq m'(151)$. Now, consider the scoring function

$$f(x, y) = x^T y - \frac{1}{k} \sum_{i=0}^{k} m'(i).$$

Denoting the function $\text{top}(k, A)$, the inputs of which are and integer k and a set of elements A, which returns the top k elements of A, we may write $f(x, y)$ as

$$f(x, y) = x^T y - \frac{1}{k} \sum_{a \in \text{top}(k,A)} a,$$

where $A = \{\cup_{\tau=-75}^{75}\ x^T y_\tau\}$. In this case, the set $\text{top}(k, A)$ and the correponsing shifts for the top k dot-products may vary for each $x \in X$ so that $f(x, y)$ is not linear with respect to the theoretical spectra. Thus, we must compute each individual dot-product for each $x \in X$. Naively, this will require $\mathcal{O}(n^2|X|)$ time, which prohibits use of $f(x, y)$ in a practical setting.

We present several algorithms which significantly reduce the compute time for calculating each individual dot-product for all $x \in X$.

Let $\mathcal{T}$ be the set of shifts of interest (for XCorr, $\mathcal{T} = \{-75, -74, \ldots, 75\}$), n_x^* be the maximum number of nonzero elements amongst all vectors $x \in X$, and n_y be the number of nonzero elements of y. For $x \in X$, let $\tilde{x}$ be the vector of tuples of the nonzero elements of x (i.e., a sparse representation), such that the first element of each tuple is a nonzero element's index in x and the second element of each tuple is the nonzero element's value in x. Denote the length of $\tilde{x}$ as n_x. Let $\tilde{y}$ similarly be a sparse represenation vector of tuples representing the nonzero elements of y whose length is n_y.

C.1 Compute all individual sliding dot-products in $\mathcal{O}(n_x^* n_y |X|)$ compute

Using Algorithm 3, we may leverage these sparse representations to compute all individual dot-products for all $x \in X$ in $\mathcal{O}(n_x^* n_y |X|)$ time. Returned is a vector A the indices of which are the the shifts of the sliding dot-product and the elements of which are the corresponding shifted dot-product. Critical to this algorithm is utilizing the fact that for elements $\tilde{y}(i,1), \tilde{x}(j,1)$, which are the indices of the ith and jth nonzero elements of x and y, respectively, they will only be multiplied and added to the dot-product of shift $(\tilde{y}(i,1) - \tilde{x}(j,1)) \bmod |\mathcal{T}|$.

C.2 Compute all individual sliding dot-products in $\mathcal{O}(m|\mathcal{T}||X|)$ compute

Let $m = \min\{n_x^*, n_y\}$. Algorithm 4 computes all individual dot-products for all $x \in X$ in $\mathcal{O}(m|\mathcal{T}||X|)$ time, running faster than Algorithm 3 when $|\mathcal{T}| < m$.

C.3 Precaching for fewest computations possible

Algorithm 6 computes all individual dot-products with the absolute fewest number of operations possible via precaching. For a theoretical peak index i, we precache all nonzero observed peaks for which i would multiply with in any dot-product. We thus avoid considering $l \notin \mathcal{T}$, as is checked in Algorithm 3. Let M be a hash function whose keys correspond to theoretical peak indices and which returns a vector of pairs corresponding to the nonzero oberved

Algorithm 3 Calculate $A = \{\cup_{\tau \in \mathcal{T}} x^T y_\tau\}$ in $\mathcal{O}(n_x^* n_y |X|)$ time

Initialize the length $|\mathcal{T}|$ vector A to all zeros.
for $i = 0, \ldots, n_y - 1$ **do**
    for $j = 0, \ldots, n_x - 1$ **do**
        $l = \tilde{y}(i,1) - \tilde{x}(j,1)$; // $\tilde{y}(i,2)$, $\tilde{x}(j,2)$ are multiplied for $\tau = \tilde{y}(i,1) - \tilde{x}(j,1)$
        if $l \in \mathcal{T}$ **then** // for continuous ranges, sufficient to check $\min\{\mathcal{T}\} \leq l \leq \max\{\mathcal{T}\}$
            $l = l \bmod |\mathcal{T}|$;
            $A(l)$ += $\tilde{y}(i,2) * \tilde{x}(j,2)$;
        end if
    end for
end for
Return A;

peak indices and values for all dot-products involving a theoretical peak. More formally, we precache M as described in Algorithm 5. Having prechached M, we thus compute all dot-products for a theoretical spectrum $x \in X$ as described in Algorithm 6. Denoting the maximum length of any of the vectors in M as d, the total resulting complexity, including precaching of M, is $\mathcal{O}(\kappa n_y + n_x^* d|X|)$. Note that $d \leq |\mathcal{T}|$ and, for MS/MS, $\kappa n_y \ll n_x^* d|X|$ so that our time complexity is $\mathcal{O}(n_x^* d|X|)$ (i.e., precaching M is neglible compared to total run time). In the worst case, the complexity of Algorithm 6 is no better than Algorithms 3 and 4, but is relatively much simpler and lends itself to efficient optimization by compilers.

Algorithm 4 Calculate $A = \{\cup_{\tau \in \mathcal{T}} x^T y_\tau\}$ in $\mathcal{O}(m|\mathcal{T}||X|)$

Initialize the length $|\mathcal{T}|$ vector A to all zeros.
if $n_y < n_x$ **then**
    $t = \tilde{y}$;
    $\tilde{y} = \tilde{x}$;
    $\tilde{x} = t$;
end if
for $i = 0, \ldots, n_x - 1$ **do**
    for $\tau \in \mathcal{T}$ **do**
        $j = i - \tau$;
        if $j \geq 0$ **then**
            $l = (\tilde{y}(i, 1) - \tilde{x}(j, 1)) \bmod |\mathcal{T}|$;
            $A(l)$ += $\tilde{y}(i, 2) * \tilde{x}(j, 2)$;
        end if
    end for
end for
Return A;

Algorithm 5 Precaching all sliding dot-products

Let maximum theoretical peak index be κ.
Initialize array of arrays M.
for $i = 0, \ldots, \kappa$ **do**
    for $j = 0, \ldots, n_y$ **do**
        $l = \tilde{y}(i, 1) - \tilde{x}(j, 1)$;
        if $l \in \mathcal{T}$ **then**
            $l = l \bmod |\mathcal{T}|$;
            $M(i)$.append$((l, \tilde{y}(j, 2)))$
        end if
    end for
end for
Return M;

Algorithm 6 Calculate $A = \{\cup_{\tau \in \mathcal{T}} x^T y_\tau\}$ with precached M

Initialize the length $|\mathcal{T}|$ vector A to all zeros.
for $i = 0, \ldots, n_x - 1$ **do**
    for $(j, v) \in M(i)$ **do**
        $A(j)$ += $v * \tilde{x}(i, 2)$;
    end for
end for
Return A;

Appendix D

CONVEX AND CONCAVE LOG-POSTERIORS FOR EFFICIENT PARAMETER ESTIMATION

We consider learning parameterized log-posteriors of the form $-\log \sum_i c_i f_i(\theta_i)$. Specifically, we are interested when these log-posteriors are concave or convex in the parameters to be learned, rendering learning tractable. One such function which arises naturally, the log-sum-exp (LSE) function, $\log \sum_i e^{x_i}$ for $x_i \in \mathcal{R}$, is widely used in machine learning for parameter estimation due to its convexity.

Let $E = e$ be a set of evidential random variables and H be a set of hidden random variables. Often of interest in Bayesian statistics is the log-posterior probability $\log p(H|E = e) = \log \frac{p(H,e)}{p(e)} = \log \frac{p(H,e)}{\sum_{h \in H} p(H=h,e)} = \log \frac{p(H,e)}{\sum_{h \in H} p(h)p(e|h)}$. We assume $p(h)$ and $p(e|h)$ are non-negative probability distributions. We assume, as is the case in Didea log-posterior scoring function (Section 6.2), that there is a particular hypothesis of hidden variables, $\tilde{h}$, for which we are interested in computing the posterior $\log p_\theta(\tilde{h}|e)$, which is parameterized θ which would like to learn. We further assume that θ is only involved in the conditional distribution of the evidence given a hypothesis of random variables (as is the case when employing virtual evidence, described in Section 5.2), so we have

$$\log p_\theta(\tilde{h}|e) = -\log \frac{\sum_{h \in H} p(h)p_\theta(e|h)}{p(\tilde{h})p_\theta(e|\tilde{h})} = -\log \sum_{h \in H} \frac{p(h)}{p(\tilde{h})} \frac{p_\theta(e|h)}{p_\theta(e|\tilde{h})} = -\log \sum_{h \in H} c_h f_h(\theta_h), \quad \text{(D.1)}$$

where c_h are constants with regards to the parameters of interest and we have parameterized function involving the evidence (one per hypothesis) $f_h(\theta_h)$. By the non-negativity of $p(h)$ and $p(e|h)$, c_h and $f_h(\theta_h)$ are both non-negative. We assume that $f_h(\cdot)$ is smooth on $\mathcal{R}$ for all $h \in H$. Note that, assuming a graphical model describing the factorization of the joint distribution, the quantities $p(h)$ and $p(e|h)$ may be efficiently computed for any given h and e. We interchangeably use n and $|H|$. In the most general setting, we assume that there is a

parameter θ_h to be learned for every hypothesis of latent variables, though if we have fewer parameters to estimate, parameter estimation becomes strictly easier.

D.1 Convex minimization

We have the following theorem describing when this quantity is concave and thus may be minimized.

Theorem 1. *The log-posterior* $\log p_\theta(\tilde{h}|e) = -\log \sum_i c_i f_i(\theta_i)$*, where* $c_i \in \mathcal{R}_+$*, is convex when each* $f_i : \mathcal{R} \to \mathcal{R}_+$ *is concave.*

Proof. By the scalar composition of convex functions [11], $d(g(x)) = d(\sum_i c_i x_i) = -\log \sum_i c_i x_i$, where $\mathcal{R}_+$ is the set of nonnegative reals and $x \in \mathcal{R}_+^n$, is convex since $g(x) = c^T x = \sum_i c_i x_i$ is concave and $d(\cdot)$ is convex, nonincreasing on $\mathcal{R}_+$. Thus, the composition $a(x) = d(g(x))$ is convex and nonincreasing in each argument $x_i \in \mathcal{R}_+$.

By the vector composition of convex functions, $a(b(x)) = a(b_1(x_1), \ldots, b_n(x_n))$ is convex when each $b_i : \mathcal{R} \to \mathcal{R}_+$ is concave. □

D.2 Concave maximization

Many applications will wish to maximize $\log p_\theta(\tilde{h}|e)$ with respect to θ, i.e.,

$$\theta^* = \operatorname*{argmax}_\theta \log p_\theta(\tilde{h}|e). \tag{D.2}$$

We are interested in determining when the quantity in Equation D.2 is concave, thus affording efficient optimization.

When $c_h = 1, f_h(\theta_h) = e^{\theta_h}$, we have the negative of the familiar convex function log-sum-exp (LSE). In order to ensure concavity of Equation D.1, it is necessary and sufficient that $\nabla_\theta^2 \log p_\theta(\tilde{h}|e) \preceq 0$. We thus have the following

$$\nabla_\theta \log p_\theta(\tilde{h}|e) = \frac{-1}{\sum_h c_h f_h(\theta_h)} \begin{bmatrix} c_1 f_1'(\theta_1) \\ \vdots \\ c_{|H|} f_{|H|}'(\theta_{|H|}) \end{bmatrix}.$$

Letting $Z = \sum_h c_h f_h(\theta_h)$, we have

$$\frac{\delta \log p_\theta(\tilde{h}|e)}{\delta\theta_i \delta\theta_j} = -\begin{cases} \frac{c_i f_i''(\theta_i) Z - (c_i f_i'(\theta_i))^2}{Z^2} & \text{if } i = j \\ \frac{-c_i f_i'(\theta_i) c_j f_j'(\theta_j)}{Z^2} & \text{if } i \neq j \end{cases}$$

Letting $a = \begin{bmatrix} c_1 f_1''(\theta_1) & \dots & c_{|H|} f_{|H|}''(\theta_{|H|}) \end{bmatrix}^T$ and $b = \begin{bmatrix} c_1 f_1'(\theta_1) & \dots & c_{|H|} f_{|H|}'(\theta_{|H|}) \end{bmatrix}^T$, we may thus write

$$\nabla_\theta^2 \log p_\theta(\tilde{h}|e) = \frac{-\text{diag}(a)}{Z} + \frac{1}{Z^2} bb^T. \tag{D.3}$$

Equation D.3 is non-negative definite if and only if, for all $x \in \mathcal{R}^n$,

$$\begin{aligned} x^T \nabla_\theta^2 \log p_\theta(\tilde{h}|e) x \leq & 0 \\ Z^2 x^T \nabla_\theta^2 \log p_\theta(\tilde{h}|e) x \leq & 0 \\ x^T(-Z\text{diag}(a) + bb^T)x \leq & 0 \\ x^T bb^T x \leq & x^T Z \text{diag}(a) x \\ (\sum_i c_i f_i'(\theta_i) x_i)^2 \leq & (\sum_i c_i f_i''(\theta_i) x_i^2)(\sum_i c_i f_i(\theta_i)). \end{aligned} \tag{D.4}$$

Determining families of functions which satisfy Equation D.4 for all $x \in \mathcal{R}^n$ is nontrivial. However, we provide some conditions for when this is true and describe the class of functions containing the LSE which are a solution to Equation D.4. We provide conditions for the homogeneous case where the function is Lipschitz, has bounded domain, and lower-bounded second derivative.

Note that, if we have a solution $g(\theta) = d((f_1(\theta_1), \dots, f_n(\theta_n)) = -\log \sum_i f_i(\theta_i)$ to Equation D.4, then the composition $g(\phi(\theta)) = g(\phi_1(\theta_1), \dots, \phi_n(\theta_n)$ is convex for all ϕ_i convex, by the vector composition of convex functions.

D.2.1 Homogenous, Lipschitz case with lower-bounded second derivative

Theorem 2. *Assume $f(\cdot) = f_i(\cdot)$ for $i = 1, \dots, n$ such that $f : [0, b] \to \mathcal{R}_+$ and $f'' \geq \bar{b}$. Assume f is Lipschitz continuous with Lipschitz constant k. Then the log-posterior $\log p_\theta(\tilde{h}|e) = -\log \sum_i c_i f(\theta_i)$ is concave when $\frac{\bar{b}(f(0) + f'(0)b + \frac{1}{2}\bar{b}b^2)}{k^2} \geq 1$.*

Proof. For the left-hand side of Equation D.4, by Lipschitz continuity, we have

$$(\sum_i c_i f'(\theta_i)x_i)^2 \leq k^2(\sum_i c_i x_i)^2$$

where

$$\begin{aligned} k^2(\sum_i c_i x_i)^2 =& k^2(\sum_i \frac{p(i)}{p(\tilde{h})}x_i)^2 \\ =& \frac{k^2}{p^2(\tilde{h})}(\sum_i p(i)x_i)^2. \end{aligned}$$

By the fundamental theorem of Calculus and the lower bound on the second derivative, we have

$$\begin{aligned} f(y) =& f(0) + \int_0^y (f'(0) + \int_0^t f''(s)ds)dt \\ \geq& f(0) + \int_0^y (f'(0) + \int_0^t \bar{b}ds)dt \\ \geq& f(0) + f'(0)y + \frac{1}{2}\bar{b}y^2 \\ \Rightarrow f(y) \geq& f(0) + f'(0)b + \frac{1}{2}\bar{b}b^2 \end{aligned}$$

Let $\beta = f(0) + f'(0)y + \frac{1}{2}\bar{b}b^2$. For the right-hand side of Equation D.4, we thus have

$$\begin{aligned} (\sum_i c_i f_i''(\theta_i)x_i^2)(\sum_i c_i f_i(\theta_i)) \geq& (\sum_i c_i \bar{b} x_i^2)(\sum_i c_i \beta) \\ \geq& \frac{\bar{b}\beta}{p^2(\tilde{h})}(\sum_i p(i)x_i^2)(\sum_i p(i)) \\ \geq& \frac{\bar{b}\beta}{p^2(\tilde{h})}(\sum_i p(i)x_i^2). \end{aligned}$$

Thus, by Jensen's inequality,

$$\begin{aligned} (\sum_i c_i f'(\theta_i)x_i)^2 \leq& \frac{k^2}{p^2(\tilde{h})}(\sum_i p(i)x_i)^2 \\ \leq& \frac{k^2}{p^2(\tilde{h})}\sum_i p(i)x_i^2. \end{aligned}$$

When $\bar{b}\beta \geq k^2$, we have

$$\begin{aligned}\frac{k^2}{p^2(\tilde{h})}\sum_i p(i)x_i^2 \leq & \frac{\bar{b}\beta}{p^2(\tilde{h})}\sum_i p(i)x_i^2 \\ \leq & (\sum_i c_i f_i''(\theta_i)x_i^2)(\sum_i c_i f_i(\theta_i)).\end{aligned}$$

□

Theorem 2 provides several insights into the values that b may take on while ensuring concavity. Let $g(b) = (f(0) + f'(0)b + \frac{1}{2}\bar{b}b^2) - \frac{k^2}{\bar{b}}$.

Corollary 2.1. *When $\bar{b} > 0$ and $(f'(0))^2 \leq 2\bar{b}f(0) - k^2$, the homogeneous log-posterior $\log p_\theta(\tilde{h}|e) = -\log \sum_i c_i f(\theta_i)$ is concave for all $0 < b < \infty$.*

Proof. From Theorem 2, $\log p_\theta(\tilde{h}|e)$ is concave when $g(b) \geq 0$. When $\bar{b} > 0$, $g(b)$ is concave-up and, when $(f'(0))^2 \leq 2\bar{b}f(0) - k^2$, has no real roots so that it is always positive. □

Corollary 2.2. *Let $A = \{\frac{-f'(0) \pm \sqrt{(f'(0))^2 - 2\bar{b}(f(0) - \frac{k^2}{\bar{b}})}}{\bar{b}}\}$. Assume $(f'(0))^2 \geq 2\bar{b}f(0) - k^2$. When $\bar{b} > 0$, the homogeneous log-posterior $\log p_\theta(\tilde{h}|e) = -\log \sum_i c_i f(\theta_i)$ is concave when $0 \leq b \leq \max\{0, \min A\}$ or $\max\{0, \max A\} \leq b \leq \infty$, and when $\bar{b} < 0$, $p_\theta(\tilde{h}|e)$ is concave when $\max\{0, \min A\} \leq b \leq \min\{0, \max A\}$.*

Proof. In this case, $g(b)$ has two real roots. By Theorem 2, when $\bar{b} > 0$, $g(b)$ is concave-up and nonnegative for all b less than or equal too its minimal root and all b greater than or equal to its maximal root. When $\bar{b} < 0$, $g(b)$ is concave-down and nonegative for all b lying between its two roots (inclusive). □

D.2.2 Generalized LSE solution

Letting l, u, v be vectors with components $l_i = x_i\sqrt{c_i f_i''(\theta_i)}, u_i = x_i\frac{c_i f_i'(\theta_i)}{\sqrt{c_i f_i(\theta_i)}}, v_i = \sqrt{c_i f_i(\theta_i)}$, we require

$$(u^T v)^2 \leq (l^T l)(v^T v). \tag{D.5}$$

Note that, by the non-negativity of c_h and $f_h(\theta_h)$, v_i is real and the quantify $l^T l$ is always real. The bound in Equation D.5 is guaranteed to hold when $l = u$ (by the Cauchy-Schwarz inequality). Thus, the log-posterior is concave when

$$\begin{aligned} l_i =& u_i \\ \Rightarrow x_i\sqrt{c_i f_i''(\theta_i)} =& x_i \frac{c_i f_i'(\theta_i)}{\sqrt{c_i f_i(\theta_i)}} \\ \Rightarrow f_i''(\theta_i) f_i(\theta_i) =& (f_i'(\theta_i))^2. \end{aligned} \tag{D.6}$$

Equation D.6 is an autonomous, second-order, nonlinear ordinary differential equation (ODE), the solution of which leads us to the following generalization of the commonly encountered LSE convex function.

Theorem 3. *$-\log \sum c_i f_i(\theta_i) = -\log \sum \alpha_i e^{\beta_i \theta_i}$ is concave with hyperparameters α_i and β_i, of which the LSE is a particular instance with $c_i = \alpha_i = \beta_i = 1$. By existence and uniqueness, all solutions to Equation D.6 must be either of this form or linear combinations of these functions.*

Proof. To simplify notation, let $t = \theta_i$ and $y(t) = f_i(t)$, where we drop the independent variable when it is understood. Thus, we are looking for y such that

$$y'' = \frac{(y')^2}{y}. \tag{D.7}$$

Let $v(t) = y'(t)$, $w(y) = v(t(y))$, and $\dot{w} = \frac{dw}{dy}$. We thus have

$$\begin{aligned} v' =& \frac{v^2}{y}, \\ \dot{w}(y) =& \frac{dw}{dy}(y) = \frac{dv}{dt}\frac{dt}{dy}\bigg|_{t(y)} = \frac{v'}{y'}\bigg|_{t(y)} = \frac{v'}{v}\bigg|_{t(y)}. \end{aligned}$$

Using Equation D.7, we have

$$\dot{w}(y) = \frac{v'(t(y))}{w(y)} = \frac{v^2(t(y))}{w(y)y} = \frac{w^2(y)}{w(y)y} = \frac{w(y)}{y}.$$

This ODE is easily solved using seperation of variables, so that we have

$$\begin{aligned}\ln w &= \ln y + c_0 \\ \Rightarrow w &= \exp(\ln y + d_0) = e^{d_0} y = d_1 y,\end{aligned}$$

where d_0 is a constant of integration. To solve for y, we have

$$w(y(t)) = v(t) = y' = d_1 y.$$

As before, $y' = d_1 y$ is easily solved using seperation of variables, giving us

$$\begin{aligned}\ln y &= d_1 t + d_2 \\ \Rightarrow y(t) &= d_3 e^{d_1 t},\end{aligned}$$

where d_1 and d_3 are constants determined by initial conditions. Returning to our earlier notation, the solution to Equation D.6 is thus $f_i(\theta_i) = d_3 e^{d_1 \theta_i}$. Letting $\beta_i = d_1$ and $\alpha_i = c_i d_3$ completes the proof. □

D.3 Concave extensions for Didea

Recall the discussion in Section 8.2. We define further extensions of Didea's learning framework which remain concave and thus afford efficient parameter learning. Didea is a special model, in that we may reparameterize its conditional log-likelihood either in terms of weighting each individual shift of the observed spectrum or we may reparameterize its conditional log-likelihood by building upon the scalar product between the observed and theoretical spectra to learn a distance matrix between the two.

D.3.1 Sparse weight vector of spectrum shifts

Building upon the reparameterization in Equation 8.9, where we learn a dense vector of weights for each shift of the observed spectrum, we describe learning a sparse vector of weights for each shift. The concave objective function of interest is then

$$\max_{\boldsymbol{\lambda}} \quad -\sum_{i=1}^{N} \log \sum_{\tau_0^i=-M}^{M} e^{\lambda_{\tau_0^i} z^i(\tau_0^i)} - \alpha \|\boldsymbol{\lambda}\|_1 = \max_{\boldsymbol{\lambda}} -\sum_{i=1}^{N} \log \sum_{\tau_0^i=-M}^{M} e^{\lambda_{\tau_0^i} z^i(\tau_0^i)} - \alpha \sum_i |\lambda_i|,$$

where $\alpha \in [0, 1]$ is the regularization strength (typically learned via cross-validation). This objective remains concave as the negative of the regularizer, $\|\cdot\|_1$, is concave.

Regularization is a common technique in machine learning, where a regularizer is added to the objective function so that solutions tend towards a particular form. In the case of L1 regularization, solutions tend to be sparse, i.e., many of the learned parameters tend to be zero. This is useful in applications where a small subset of parameters are better at explaining the underlying better than a dense set of parameters. In the case of Didea (and possibly XCorr), such regularization would be helpful if a compact set of observed spectrum shifts were better at discriminating against random peptide matches then the entire set of observed spectrum shifts.

D.3.2 Distance metrics between the observed and theoretical spectra

We describe an extension of Didea's learning framework to learn different distance metrics between the theoretical and observed spectra. Specifically, we learn a distance matrix, A, between the theoretical. Recall that Didea's original log-likelihood was

$$\begin{aligned}
\log p(\tau_0 = 0|s, x, \lambda) &= \log e^{\lambda(b+y)^T s} - \log \sum_{\tau=-M}^{M} e^{\lambda(b+y)^T s_\tau} \\
&= -\log \sum_{\tau=-M}^{M} e^{\lambda[(b+y)^T s_\tau - (b+y)^T s)]} \\
&= -\log \sum_{\tau=-M}^{M} e^{\lambda(b+y)^T (s_\tau - s)} \\
&= -\log \sum_{\tau=-M}^{M} e^{\lambda(b+y)^T \hat{s}_\tau}.
\end{aligned}$$

We reparameterize this as

$$\log p(\tau_0 = 0|s, x, \lambda) = -\log \sum_{\tau=-M}^{M} e^{(b+y)^T A \hat{s}_\tau}, \tag{D.8}$$

where $A \in \mathcal{R}^{2M+1} \times \mathcal{R}^{2M+1}$. Equation D.8 is concave, as it is a linear function of A so that, by the convexity of the LSE and the composition rules for convex functions, $\log \sum_{\tau=-M}^{M} e^{(b+y)^T A \hat{s}_\tau}$

remains convex in A (we may also rewrite $(b + y)^T A\hat{s}_\tau = l^T A s'$ as $z^T \alpha$, where $z = [l_1 s'_1 l_1 s'_2 \dots l_1 s'_n \; l_2 s'_1 \; l_2 s'_2 \; \dots \; l_2 s'_n \; l_3 s'_1 \; \dots \; l_n s'_n]$ and $\alpha = [a_{11} \; \dots \; a_{1n} \; a_{21} \; \dots \; a_{2n} \; \dots \; a_{nn}]$, in which case it is obvious Equation D.8 is concave in α and thus A). The overall learning objective is then

$$\max_A - \sum_{i=1}^{N} \log \sum_{\tau=-M}^{M} e^{(b^i+y^i)^T A\hat{s}^i_\tau}. \tag{D.9}$$

In order to derive gradients for Equation D.9 so that we may optimize it using stochastic steepest ascent, we rewrite the objective function as

$$\begin{aligned}
&\max_A - \sum_{i=1}^{N} \log \sum_{\tau=-M}^{M} e^{(b^i+y^i)^T A\hat{s}^i_\tau} \\
&\max_A - \sum_{i=1}^{N} \log \sum_{\tau=-M}^{M} e^{(l^i)^T A\hat{s}^i_\tau} \\
&\max_A - \sum_{i=1}^{N} \log \sum_{\tau=-M}^{M} \exp \sum_j l^i_j \sum_k a_{jk}\hat{s}^i_{\tau,k},
\end{aligned}$$

where $l^i = b^i + y^i$, a_{jk} is the jth row and kth column element of A, and $\hat{s}^i_{\tau,k}$ is the kth element of vector $\hat{s}^i_\tau$. Letting $f(A) = -\log \sum_{\tau=-M}^{M} e^{(b^i+y^i)^T A\hat{s}^i_\tau}$, we thus have

$$\begin{aligned}
\frac{\delta f(A)}{\delta A_{jk}} &= \frac{-1}{\sum_\tau e^{(l^i)^T A\hat{s}^i_\tau}} \sum_\tau l^i_j \hat{s}^i_{\tau,k} e^{(l^i)^T A\hat{s}^i_\tau} \\
\Rightarrow \frac{\delta f(A)}{\delta A} &= \frac{-1}{\sum_\tau e^{(l^i)^T A\hat{s}^i_\tau}} \sum_\tau l^i (\hat{s}^i_\tau)^T e^{(l^i)^T A\hat{s}^i_\tau}.
\end{aligned} \tag{D.10}$$

The SGA update in Equation D.10 has a very intuitive form. A is updated as a linear combination of the outer products between l^i and $\hat{s}^i_\tau$ for all shifts τ. This outer product, as well as the matrix product $(l^i)^T A\hat{s}^i_\tau$, may be efficiently computed using optimized linear algebra packages.

Appendix E

MS/MS DATABASE SEARCH TIMING RESULTS

Table E.1: % running time of trellis compared to XCorr mixture model.

dataset	*trellis*	XCorr
Yeast-1	9.98%	100%
Worm-1	6.17%	100%
Malaria	10.56%	100%

Table E.2: % running time of trellis methods compared to original DRIP.

dataset	$trellis_{speed}$	$trellis_{base}$	DRIP
Yeast-1	7.78%	14.80%	100%
Worm-1	8.59%	16.36%	100%
Malaria	8.05%	15.50%	100%