Computational Methods for Protein Inference in Shotgun Proteomics Experiments

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

M. Sc. Julianus Pfeuffer

aus Werneck

Tübingen

2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	15.11.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Oliver Kohlbacher
2. Berichterstatter:	Prof. Dr. Manfred Claassen

Erklärung

Ich erkläre hiermit, dass ich die zur Promotion eingereichte Arbeit mit dem Titel:

Computational Methods for Protein Inference in Shotgun Proteomics Experiments

selbständig verfasst, nur die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche gekennzeichnet habe. Ich erkläre, dass die Richtlinien zur Sicherung guter wissenschaftlicher Praxis der Universität Tübingen (Beschluss des Senats vom 25.5.2000) beachtet wurden. Ich versichere an Eides statt, dass diese Angaben wahr sind und dass ich nichts verschwiegen habe. Mir ist bekannt, dass die falsche Abgabe einer Versicherung an Eides statt mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft wird.

Abstract

Since the beginning of this millenium, the advent of high-throughput methods in numerous fields of the life sciences led to a shift in paradigms. A broad variety of technologies emerged that allow comprehensive quantification of molecules involved in biological processes. Simultaneously, a major increase in data volume has been recorded with these techniques through enhanced instrumentation and other technical advances. By supplying computational methods that automatically process raw data to obtain biological information, the field of bioinformatics plays an increasingly important role in the analysis of the ever-growing mass of data. Computational mass spectrometry in particular, is a bioinformatics field of research which provides means to gather, analyze and visualize data from high-throughput mass spectrometric experiments. For the study of the entirety of proteins in a cell or an environmental sample, even current techniques reach limitations that need to be circumvented by simplifying the samples subjected to the mass spectrometer. These pre-digested (so-called bottom-up) proteomics experiments then pose an even bigger computational burden during analysis since complex ambiguities need to be resolved during protein inference, grouping and quantification.

In this thesis we present several developments in the pursuit of our goal to provide means for a fully automated analysis of complex and large-scale bottom-up proteomics experiments.

Firstly, due to prohibitive computational complexities in state-of-the-art Bayesian protein inference techniques, a refined, more stable technique for performing inference on sums of random variables was developed to enable a variation of standard Bayesian inference for the problem.

Based upon the integration of this technique into a co-developed inference library, an implementation of an OpenMS tool for Bayesian protein inference is then presented. It performs efficient inference on discrete Bayesian networks through loopy belief propagation. Despite handling the protein inference problem in a rigorously probabilistic manner and therefore achieving excellent sensitivity on ground-truth data, it outperforms most of the state-of-the-art techniques in computational efficiency. The tool's interface provides unique features for in- and output and options for regularization of proteins in groups to tweak calibration.

Lastly, a complete, easy-to-use, and scalable workflow for protein inference and quantification was built around the new tool. The pipeline is implemented in *nextflow* and part of a set of standardized, well-tested, and community-maintained workflows by the *nf-core* collective. Our workflow runs on large scale data with complex experimental designs and allows a one-command analysis of local and publicly available data sets with state-of-the-art accuracy on various high-performance computing environments or the cloud.

Kurzfassung

In den letzten Jahrzehnten kam es zu einem signifikanten Anstiegs des Einsatzes von Hochdurchsatzmethoden in verschiedensten Bereichen der Naturwissenschaften, welche zu einem regelrechten Paradigmenwechsel führte. Eine große Anzahl an neuen Technologien wurde entwickelt um die Quantifizierung von Molekülen, die in verschiedenste biologische Prozesse involviert sind, voranzutreiben und zu beschleunigen. Damit einhergehend konnte eine beträchtliche Steigerung an Daten festgestellt werden, die durch diese verbesserten Methoden generiert wurden. Durch die Bereitstellung von computergestützten Verfahren zur Analyse eben dieser Masse an Rohdaten, spielt der Forschungsbereich der Bioinformatik eine immer größere Rolle bei der Extraktion biologischer Erkenntnisse. Im Speziellen hilft die computergestützte Massenspektrometrie bei der Prozessierung, Analyse und Visualisierung von Daten aus massenspektrometrischen Hochdursatzexperimenten. Bei der Erforschung der Gesamtheit aller Proteine einer Zelle oder einer anderweitigen Probe biologischen Materials, kommen selbst neueste Methoden an ihre Grenzen. Deswegen greifen viele Labore zu einer, dem Massenspektrometer vorgeschalteten, Verdauung der Probe um die Komplexität der zu messenden Moleküle zu verringern. Diese sogenannten „Bottom-up“-Proteomikexperimente mit Massenspektrometern führen allerdings zu einer erhöhten Schwierigkeit bei der anschließenden computergestützen Analyse. Durch die Verdauung von Proteinen zu Peptiden müssen komplexe Mehrdeutigkeiten während Proteininferenz, Proteingruppierung und Proteinquantifizierung berücksichtigt und/oder aufgelöst werden.

Im Rahmen dieser Dissertation stellen wir mehrere Entwicklungen vor, die dabei helfen sollen eine effiziente und vollständig automatisierte Analyse von komplexen und umfangreichen „Bottom-up“-Proteomikexperimenten zu ermöglichen.

Um die hinderliche Komplexität diskreter, Bayes’scher Proteininferenzmethoden zu verringern, wird neuerdings von sogenannten Faltungsbäumen (engl. „convolution trees“) Gebrauch gemacht. Diese bieten bis jetzt jedoch keine genaue und gleichzeitig numerisch stabile Möglichkeit um „max-product“-Inferenz zu betreiben. Deswegen wird in dieser Dissertation zunächst eine neue Methode beschrieben die das mithilfe eines stückweisen bzw. extrapolierendem Verfahren ermöglicht.

Basierend auf der Integration dieser Methode in eine mitentwickelte Bibliothek für Bayes’sche Inferenz, wird dann ein OpenMS-Tool für Proteininferenz präsentiert. Dieses Tool ermöglicht effiziente Proteininferenz auf Basis eines diskreten Bayes’schen

Netzwerks mithilfe eines „loopy belief propagation“ Algorithmus’. Trotz der streng probabilistischen Formulierung des Problems übertrifft unser Verfahren die meisten etablierten Methoden in Recheneffizienz. Das Interface des Algorithmus’ bietet außerdem einzigartige Eingabe- und Ausgabeoptionen, wie z.B. das Regularisieren der Anzahl von Proteinen in einer Gruppe, proteinspezifische „Priors“, oder rekalibrierte „Posteriors“ der Peptide.

Schließlich zeigt diese Arbeit einen kompletten, einfach zu benutzenden, aber trotzdem skalierenden Workflow für Proteininferenz und -quantifizierung, welcher um das neue Tool entwickelt wurde. Die Pipeline wurde in *nextflow* implementiert und ist Teil einer Gruppe von standardisierten, regelmäßig getesteten und von einer Community gepflegten Standardworkflows gebündelt unter dem Projekt *nf-core*. Unser Workflow ist in der Lage selbst große Datensätze mit komplizierten experimentellen Designs zu prozessieren. Mit einem einzigen Befehl erlaubt er eine (Re-)Analyse von lokalen oder öffentlich verfügbaren Datensätzen mit kompetetiver Genauigkeit und ausgezeichneter Performance auf verschiedensten Hochleistungsrechenumgebungen oder der Cloud.

To my beloved father Ulrich who inspired me to
start this journey but cannot see its happy end.

Acknowledgments

I would like to express my sincere gratitude to my advisors Prof. Oliver Kohlbacher and Prof. Knut Reinert for their continuous support and guidance in my research and during the writing of this thesis. Further I want to thank Prof. Oliver Serang for his time with us in Berlin, teaching us valuable lessons regarding both life in general, and all kinds of math and code (plus pop culture).

I also thank all the colleagues and collaborators that I met along my way for their fruitful discussions and interesting projects that we did together. Specifically, I want to thank those people that I meanwhile consider friends, such as Timo Sachsenberg for almost daily discussions about coding, mass spec and electronic music (despite hundreds of kilometers of distance), Alexander Fillbrunn for all the nice de.NBI trips and beautiful KNIME workflows we did together, Oliver Alka for his selfless support with all those tedious infrastructure tasks, and Yasset Perez-Riverol for the always funny but also very dedicated collaboration.

Additionally, I would like to thank all developers of OpenMS. I am grateful for all the developer and user meetings we did together and all the helpful discussions and reviews on GitHub. I bet there is at least one line of code from each of them, that made a difference in this thesis. Without the users OpenMS my dedication to this project would not have been this strong and the continuous improvements from feedback made the tools as cool as they are now.

I also would like to express my gratitude towards my long-term friends Raphael, Marko, Firat, Niko and Michaela. Without their support and diversion it would have been a boring time.

Now I want to thank my partner Verônica who was always giving me the right amount of happy distractions or kicks in the behind, whatever was needed most at a time. Meeting you was worth more than receiving this degree.

Last but most importantly, I would like to thank my family, which includes my parents Jutta, Ulrich and Michael, as well as my brother Maximilian for raising me or growing up with me and therefore paved the way to my academic career and making me who I am today. Certainly, they also gave me strength throughout the whole process of writing this thesis without which this whole endeavor probably would not have been possible.

In accordance with the standard scientific protocol, I will use the personal pronoun *we* to indicate the reader and the writer or my scientific collaborators and myself.

L

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

For the investigation of certain diseases, elucidation of the proteins present in a cell has become a crucial part in many biological and medical studies nowadays. Unlike deoxyribonucleic acid (DNA) and ribonucleic acid (RNA), the entirety of proteins present in a cell or a compartment thereof can provide a more immediate view on currently active processes. An established method to measure the abundance of proteins and their proteoforms in a sample is mass spectrometry (MS). Due to limitations in the experimental process however, usually an indirect approach is used, where proteins are digested into peptides first (also known as bottom-up proteomics experiments). This leads to ambiguities in the reverse process of mapping the peptides back to their proteins of origin—the so-called protein inference problem[108]. The problem is aggravated by the fact that the identification process of peptides from mass spectra carries uncertainty from the imperfections of both experimental and analytical procedures.
Ideally, a protein inference algorithm therefore fulfills the following properties:

- it recognizes and reports on uncertainty on the protein level,
- it is able to use as much information as possible (including information from ambiguous peptides),
- it neither under- nor over-counts evidence from ambiguous peptides, and
- it allows for grouping of proteins into ambiguity groups whenever the present information is not enough to make a clear distinction.

All these requirements are fulfilled by probabilistic inference on generative models that allow for an arbitrary level of accuracy in modeling the underlying experimental processes. Generative models in the form of Bayesian networks (BNs) were already shown to work well on this specific problem[163], however the tools performing inference on the current, still relatively simple models face two yet unsolved problems:

- They are prohibitively slow for large datasets with the applied techniques, and

- their models and the chosen type of inference are usually too optimistic for proteins without unique evidence[160].

Additionally, a tool that implements an efficient inference method is a requirement for a coherent, feature-complete proteomics pipeline for the analysis of large-scale MS data on high-performance computing (HPC) environment or the cloud. However, full quantitative proteomics pipelines are often either wrapped into one-tool-solutions that do not benefit from distributed hardware or are developed separately by different labs specifically for a certain computing infrastructure. The deployment and configuration of such a pipeline can be a non-negligible obstacle for other users due to the large amount of software to be installed and the often simultaneously required domain knowledge in both high-performance computing and proteomics at the same time. Lab-internal workflows are sometimes hard to get support for or lack robustness on types of data not (yet) used by the developers. Another drawback of one-tool-solutions on the other hand is that it is hard for users to exchange parts of the internal pipeline with the latest improvements from the proteomics community or to extend its functionality (with for example custom quality control tools based on intermediate results).

1.2 Outline of this thesis

Succeeding the introduction, Chapter 2 aims at familiarizing the reader with the relevant technical, biological and algorithmic background. Towards the end of the background we also review current approaches for the protein inference problem including their insufficiencies in either accuracy or efficiency. Chapters 3-5 of this thesis, then cover three advancements to the field of computational MS that help overcome these unsatisfactory properties.

1. In Chapter 3, we develop and compare several more accurate mathematical methods for efficient max-product inference on Bayesian networks with additive dependencies to overcome the limitations of current approaches with probabilistic models commonly used for protein inference. The methods extend a previously described method[139] that makes use of the fact that max-convolution involves calculating maxima of vectors, which is equivalent to an infinity norm. The infinity or Chebyshev norm can in turn be approximated by a high enough fixed p-norm. We show how to choose one or multiple values for p to gain the most accurate results without numeric instabilities and how to even further increase accuracy by affine corrections and extrapolation via a null space projection.

2. Chapter 4 describes the development of EPIFANY[119], a ready-to-use tool in the Open Mass Spectrometry (OpenMS) framework for the protein inference problem, using the results of chapter 3 together with efficient approximate loopy belief propagation inference and an improved probabilistic model that is able to distribute evidence from peptides unevenly to proteins in a rigorous Bayesian framework. We show on a complex protein standard that it is faster than all other tested probabilistic methods with comparable or better accuracy. It also scales better and even outperforms heuristic approaches on larger data. EPIFANY also offers more flexibility in input and output options, such as user-defined protein priors or output peptide posteriors.

3. In Chapter 5, we present a user-friendly pipeline that uses EPIFANY among a plethora of other tools for end-to-end analysis of label-free bottom-up proteomics MS data, ready to be used on large datasets through execution in the cloud or HPC systems. Generalizability across HPC environments and clouds is ensured—mostly hidden from the user—by its implementation in the nextflow[32] workflow framework. It is furthermore a community-driven effort, part of nf-core[42], that adheres to established guidelines regarding pipeline structure, software packaging (e.g., into containers and bioconda[50]) and continuous testing. Lastly, on an elaborate benchmarking dataset we ensured that accuracy is on par with the state-of-the-art one-tool-solution MaxQuant[27] at a better efficiency and with higher inherent scalability.

In the concluding Chapter 6, we summarize our contributions and discuss to which degree the motivating problems could be solved or where additions or improvements could still be made.

Chapter 2

Background

2.1 Protein biosynthesis

Protein biosynthesis starts with the information contained in the unwound parts of the genome encoded as DNA in the nucleus of a cell. During transcription protein coding parts of the double-stranded DNA are temporarily separated and transcribed by a group of enzymes into messenger RNA (RNA)). The resulting single-stranded sequence of ribonucleotides is either co-transcriptionally or immediately after transcription processed further by the spliceosome (in a process called splicing). During splicing, introns (non-coding regions) are removed and exons (coding regions) are joined together. Note that exon skipping events might occur in which the spliceosome cuts at the beginning of one and at the end of another downstream intron such that the resulting splice form is shorter than the unspliced (i.e., constituent) form. The final translation into proteins takes place outside the nucleus. Triplets of ribonucleotides are consecutively recognized by the ribosome, which synthesizes the resulting protein by initiating and elongating its polypeptide sequence of amino acids with the help of transfer RNA (RNA). Important events that occur during or after translation include proteolysis (which removes N-terminal, C-terminal or internal amino acid residues or peptides from the polypeptide) as well as post-translational modification (during which the termini and side-chains of the polypeptide are enzymatically modified).

2.2 Proteomics

Proteomics is defined as the large-scale study of the proteome, the entirety of proteins that is produced or modified by an organism or system. As opposed to genomics and even transcriptomics, proteomics provides a direct measure of the identities and quantities of proteins present in the studied entity. Post-transcriptional regulation can have a significant impact on protein concentrations in a cell (especially on absolute concentrations outside of a steady-state)[63,87]. The study of proteins therefore grants researchers a direct measure of the level of translation when examining cellular processes and allows for determination of post-translational functional modifications, as well as protein isoforms. Nowadays, a common method to investigate the proteome on a large scale is mass spectrometry.

2.3 Mass spectrometry-based proteomics

Most setups for mass spectrometry-based proteomics experiments consist of at least the following sequence of steps: firstly, biological samples are extracted and prepared. Then, to reduce sample complexity, a separation of the analytes is performed. Lastly, the analytes are subjected to the mass spectrometer where they are ionized. The instrument then measures mass and abundance of the ions and outputs mass spectra.

Subsequent analysis of the contained abundance and mass of measured ions using computational methods allows for identification and quantification of the original analytes.

2.3.1 Sample preparation

During sample preparation, cells need to be lysed first to release their content. To trigger an immediate stop of the current cellular processes, active enzymes are artificially inhibited. The resulting sample content is split into proteins of interest and remaining components by centrifugation and depletion. In some cases (e.g., when studying certain post-translational modifications like phosphorylations) an additional enrichment step is used to increase the concentration of proteins of specific types in the sample. Furthermore, the three-dimensional structure of proteins has to be unfolded in a denaturation step for optimal results in downstream digestion. This is done by adding chaotropic reagents as well as by reducing disulfide bonds and subsequently alkylating the resulting sulfhydryl groups. To clean the pre-processed lysate from undesirable molecules (like salts, detergents, chaotropic reagents, glycerol or buffer) dialysis and desalting techniques are applied. Since large macromolecules with potentially many charged side-chains introduce difficulties during downstream experimental and computational analysis[23,162], the resulting linearized proteins are often subjected to digestion via endoproteases (in so-called “bottom-up” or “shotgun” proteomics experiments). Digestion can be done in-solution or in-gel if a pre-fractionation is preferred (e.g., in the case of highly complex samples). Bottom-up experiments and the challenges during their analysis, will be the focus of this manuscript.

2.3.2 Separation techniques

As mentioned before, samples can be pre-fractionated using techniques such as one- or two-dimensional sodium dodecyl sulfate polyacrylamide gel electrophoresis (SDS PAGE)[76] during or before denaturation and digestion. After electrophoresis, protein bands or spots are visualized and extracted from the gel matrix and prepared for MS analysis.

Another separation technique, high-performance liquid chromatography (HPLC), is usually coupled directly to the mass spectrometer (then called LC-MS). It is realized by directing the liquified sample through a thin column comprised of a mobile and stationary phase before it reaches the mass spectrometer. During their flow through the mobile phase of the column, peptides are slowed down by interactions with the stationary phase depending on their physicochemical properties (e.g., hydrophobicity for reversed-phase LC[96]). Their retention time, the time from injection to elution, separates peptides in an additional dimension.

2.3.3 Mass spectrometry

A typical mass spectrometer consists of three principal components (Figure 2.1):

1. The ion source: the component that ionizes the analyte molecules—often realized through soft, i.e., non-degrading electrospray ionization (ESI)[55],
2. an analyzer that sorts and filters the resulting ions according to their mass-to-charge ratio (m/z), and
3. a detector that records the ions that were selected by the analyzer. The current generated by the stimulating ions in proximity or in contact with the detector is proportional to the number of ions.

Each measurement cycle at a specific retention time produces a so-called mass spectrum that comprises all pairs of detected m/z ratios and their associated ion intensity (a digitized value of the measured current). The reported intensities are proportional to the number of ions that excited the detector at this m/z value. Common mass analyzers use the behavior of an ion in an electric or magnetic field to calculate its m/z ratio. Mass-determining characteristics are for example:

- their time-of-flight in a vacuum tube (see Campana [15] for a historical overview)
- the shape and frequency of their orbit between electrodes (as with quadrupole[56] or Orbitrap[89] instruments) or around a superconducting magnet (Fourier-transform ion cyclotron resonance (FTICR)-MS[22])

Mass spectra can be stacked along the retention time dimension to create a two-dimensional representation of the resulting data, often called “peak maps” (Figure 2.2). Modern instruments are able to record dozens of spectra per second, therefore the size of a spectrum file on a computer can easily reach gigabytes.

2.3.4 Tandem mass spectrometry

Since the m/z ratio of an ion does not distinguish between isobaric peptides (i.e., peptides with the same mass), a second stage of measurements is needed in the mass spectrometer for the identification of peptide sequences. To discriminate between different amino acid sequences, peptides need to be fragmented into overlapping sequence fragments, as this allows to reconstruct the sequence based on the mass differences between them.

An ideal fragmentation technique yields complete “ion ladders” of both prefix and suffix ions through fragmentation of the peptide once around each peptide bond. Common fragmentation methods that yield more or less complete ion ladders of different types, include electron-transfer dissociation (ETD[155]), collision-induced dissociation (CID[169]), higher-energy collisional dissociation (HCD[109]) or hybrid techniques such as EthcD[44].

For fragmentation, analytes in a small isolation window around an interesting (usually a local high-intensity) “precursor” peak are trapped in a collision cell and (depending on the applied method) subjected to collision with an inert gas which breaks the sequence at covalent bonds between the α-carbon of consecutive amino acids (i.e., the peptide bond or one of its adjacent bonds). The frequencies at which exact bond the collision causes a break, depends on the used fragmentation technique.

Although multiple fragment scans can be recorded between two consecutive survey scans, precursors are chosen carefully based on predefined or real-time adapted inclusion/exclusion lists such that ideally, a single analyte is only measured once at the beginning of its elution and once at its apex (see Figure 2.2 for a typical precursor selection).

The resulting fragment ions are collected again in an analyzer and intensities for different m/z values are measured a second time. These second stage measurements then yield so-called fragment, tandem MS, MS^2, or as used in this thesis: MS/MS scans, that can now be used for identification of peptides through computational approaches (Figure 2.3).

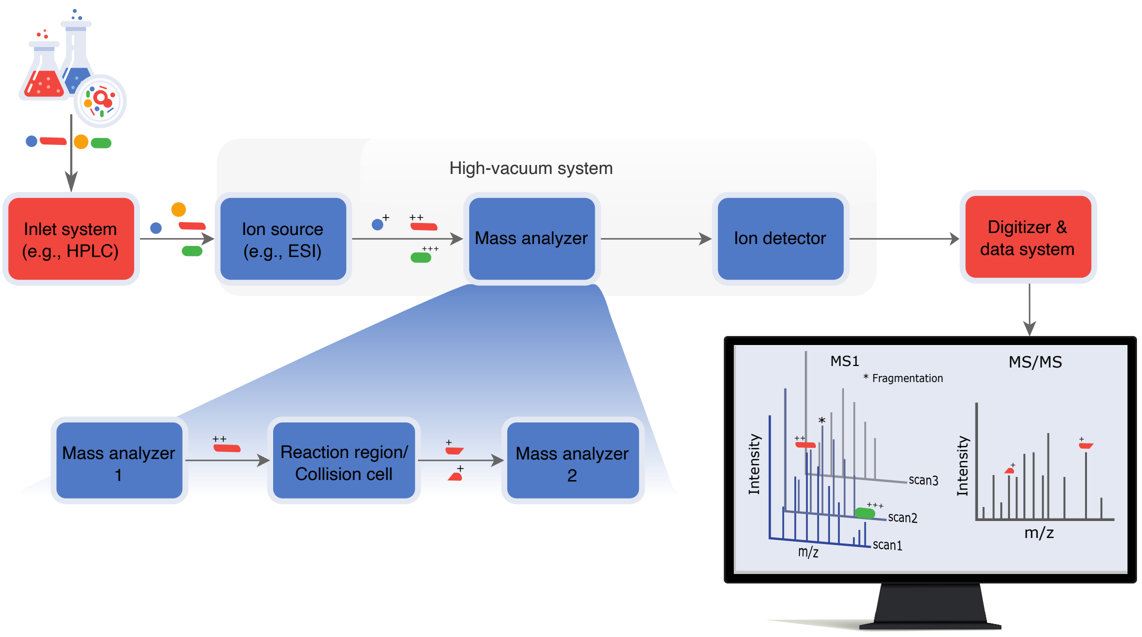


Figure 2.1: Components of a mass spectrometer conceptualized. The sample is depicted as four types of molecules, distinguished by different shapes and colors (top left). Their relative positions along the connecting arrows represents their time of arrival in the next component. Through the inlet system, parts of the sample are continuously—e.g., when an HPLC is coupled to it—or manually injected over time into the inlet of the mass spectrometer (parts in blue boxes). The ion source then generates charged particles from the arriving molecules if they are ionizable (here e.g., orange molecules are not ionizable). It then introduces them into the high-vacuum system of the mass spectrometer where the mass analyzer(s) are located. In the case of tandem MS, mass analysis is further split into: a first analyzer conducting survey/precursor scans and filtering for interesting mass ranges, a collision cell in which selected ions are being trapped for subsequent fragmentation, and a second mass analyzer (MA) generating fragment scans. In this example, ions of the red and green type reach MA1 at the same time but only red is selected for fragmentation due to the devised higher intensity of its type. The ion detector measures the ions from both analyzers and generates an electric current proportional to the number of ions measured at a certain m/z ratio. Note, that depending on the type of analyzer, detection may take place in the same compartment. The generated current is finally digitized in the data system and processed to be displayed as MS1 (for analyzer 1) and MS/MS spectra (for analyzer 2).

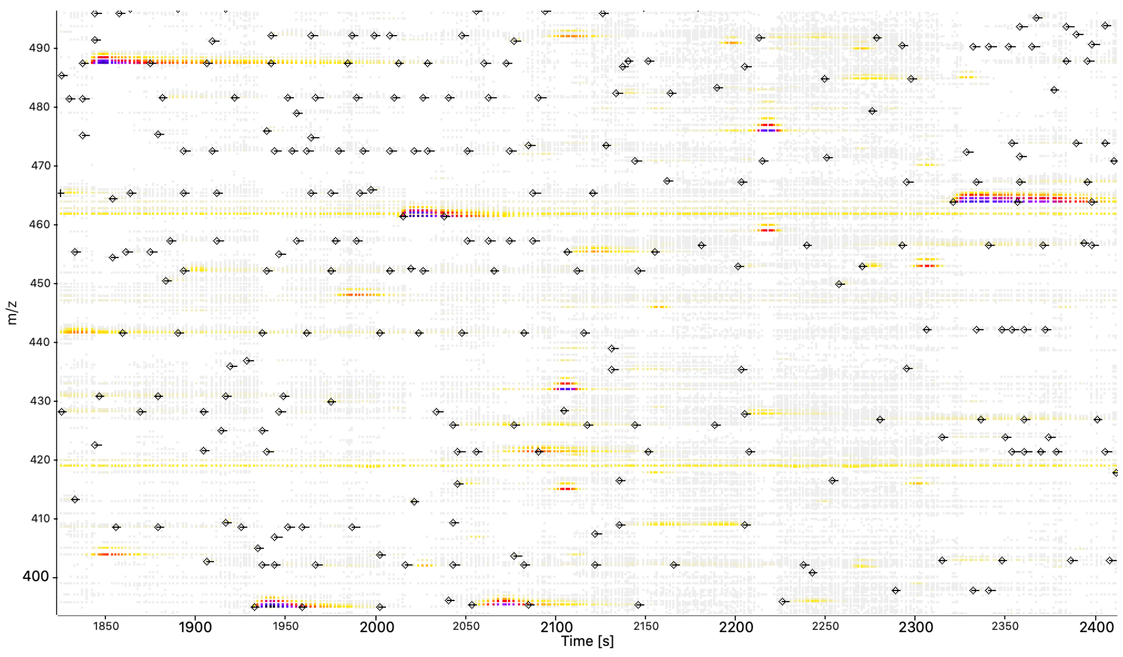


Figure 2.2: Peak Map exported from TOPPView[152] of a low-complexity sample displaying hundreds of mass spectra, stacked along the x-axis according to retention time. A few thousand peaks are plotted as small rectangles. While gray peaks mostly originate from noisy measurements, multiple analytes can be visually detected as areas of darker colors which signify a larger number of ions recorded. Little diamonds show triggered MS/MS scans for identification.

2.4 Computational mass spectrometry

Due to the rapid evolution of mass spectrometry instruments and experimental techniques, computational mass spectrometry has expanded to an independent scientific field with a variety of distinct methods and tools. Solely concepts and algorithms with immediate relevance to the contributions described in this thesis are introduced in this section. Those include computational techniques for peptide identification, protein inference, and quantification.

2.4.1 Quantification

A common goal is to quantify the measured molecules in a sample. While determining absolute concentrations is difficult without internal standards (due to different ionization efficiencies), relative quantification can be performed by simply comparing intensities of the same analytes across samples and is sufficient for a wide range of e.g., clinical studies. Firstly, the spread-out signals of analytes (and their isotopes) need to be detected in the peak maps (Figure 2.2). One possible algorithm starts at seeds in high intensity regions or at precursors of confidently identified fragment scans (see also Subsection 2.4.2) and iteratively updates and validates the shape of a three-dimensional model there. The shape is determined by the roughly Gaussian-shaped elution profile[51] along the retention time axis in the case of HPLC-MS and the expected isotope pattern along the m/z axis. The isotope pattern arises from the different relative abundances of isotopes in natural molecules. For peptides of unknown composition but known mass, a peptide composed of solely "averagine" amino acids can be assumed[134].

Since repeated HPLC-MS runs, even in temporal proximity on the same LC column, may suffer from potential retention time shifts, relative quantification requires the alignment of the located features across samples (e.g., through pose clustering approaches[77]).

Finally, after finding best matches across runs, the intensities of all peaks in a feature can be summarized and compared based on for example user-defined contrasts.

2.4.2 Peptide identification

To elucidate what was quantified, a separate identification step is necessary in a proteomics processing pipeline. It is needed to assign a peptide sequence to the features that were discovered during quantification. This is achieved indirectly based on the assignment of a peptide sequence to the MS/MS scans that were triggered inside a feature[25].

Scoring peptide-spectrum matches

The analysis of mass spectra to identify peptide sequences involves scoring an MS/MS scan based on several known or predictable characteristics (e.g., the precursor m/z, specific peaks in the MS/MS spectrum, or retention time). While precursor mass and retention time information is mainly used to roughly pre-filter candidates, scoring algorithms for the spectral composition were the main focus of research in computational mass spectrometry for many years[106]. In an abstract point of view, a scoring algorithm consists of two components:

- A reference in the form of a potential alphabet of amino acids and modifications (for so-called *de novo* identification), a database of possible sequences, or a database of previously identified spectra,
- and a scoring function to compare the recorded spectra to the reference. The type of scoring function certainly depends on the reference. For *de novo* methods[60], graph-based scoring to find the best explanation (i.e., the highest scoring path) of mass differences in an ion ladder is a common approach[18]. In the case of sequence database searches, scoring is based on the comparison of an *in silico* generated theoretical spectrum and the experimentally observed spectrum. The same spectrum similarity measures can also be applied to spectra from spectral libraries.

In the rest of this manuscript we will focus on the still most commonly used type of reference: the reference sequence database, which is summarized in Figure 2.3.

Scoring functions for peptide-spectrum matchs (PSMs) to evaluate the similarity of theoretical and experimental spectra can range from simple and fast dot product scores testing the absence or presence of certain masses[28,39,40,72], to more elaborate scores calculating likelihoods according to a probabilistic model[7,53,176]. While simple dot product measures require setting up a tolerance or binning the spectrum to account for the limited mass accuracy of instruments, some probabilistic approaches can model the mass-error as just another continuous variable. Most methods also report or even score auxiliary information like the number of consecutively matched ions of the same fragment types, i.e. the size of so-called ion ladders (Figure 2.4).

Similarly, several choices exist for the generation of theoretical spectra. Research in that area revolves around how likely certain fragmentation events are and therefore

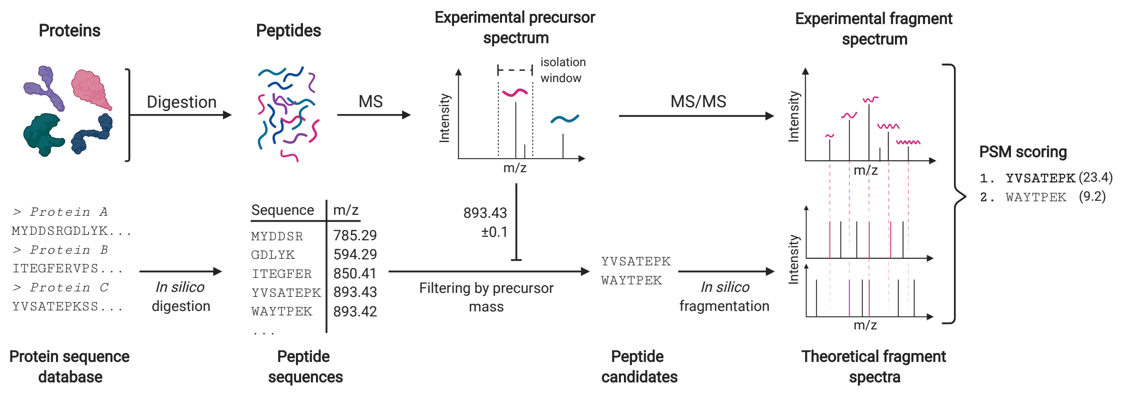


Figure 2.3: Database search as a schematic representation. Every experimental fragment spectrum (top right), is scored against a set of theoretical fragment spectra (bottom right). Experimental spectra are generated from those candidate sequences of an *in silico* digested protein database that passed a filter based on the proximity of their theoretical m/z to the experimental precursor m/z, which triggered the MS/MS scan. Both precursor filtering and peak matching usually allow for tolerances that account for different resolutions across instruments. Height and number of matched peaks have a high impact on the resulting score which is used to rank the final candidates (see the exemplary list on the far left).

which fragments to include and how to model their relative intensities. Existing algorithms can be divided into fast binary or ion-type weighted generators with strong independence assumptions included in classical search engines[28,39,40,72] and slower, highly parameterized machine learning algorithms that can model more complex dependency structures between fragment ions of different types but require lots of data for training[30,47,130,156]. Since scoring—especially with many potential modifications—is a combinatorial problem and computationally expensive, the amount of peptides to be scored can be reduced by hybrid methods using sequence tags (i.e., partial high-confidence *de novo* sequence matches) in conjuction with fast database lookup[158] or precomputed indices of theoretical fragments[75].

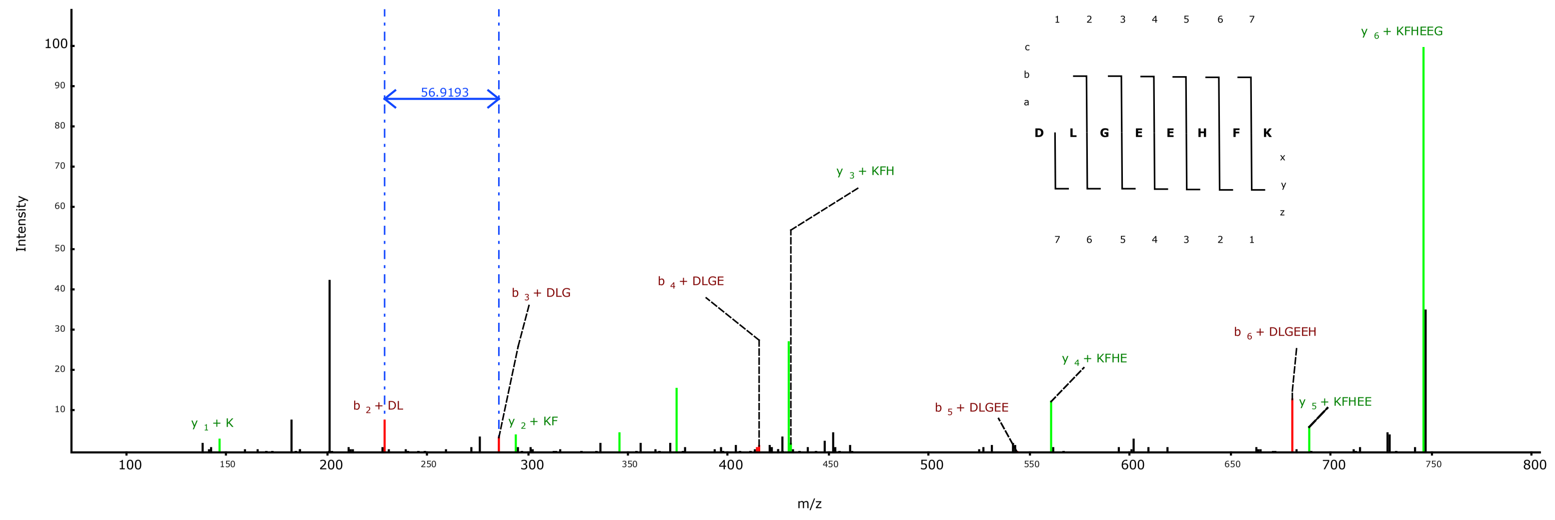


Figure 2.4: MS/MS scan as displayed in TOPPView[152]. Experimentally observed peaks (black) are matched against m/z ratios of *in silico* generated fragment ions by TOPPView with a small tolerance. The candidate sequence in question is shown in the text inlet (on the top right). Matched peaks are colored red for y-ions from the C-terminus and green for b-ions from the N-terminus. The text inset shows the candidate sequence along with the matched ions (i.e. hooks pointing to ion types). Other ion types were ignored during matching. The blue dot-dashed lines measure the distance between an exemplary pair of consecutive peaks of the same ion-type (b). The mass difference of 56.9193 indicates that the associated prefix ions differ by glycine (G) at the end of their sequence.

However, all database search methods are affected by two fundamental problems:

- By relying on a database of possible proteins and usually a set of possible modifications it can in the perfect case only identify the best peptide match in this predefined collection,

- and even with complete databases, due to noisy or incomplete spectra and imperfect scoring schemes, wrong peptide hypotheses may score better than correct ones.

Two mostly complementary ways emerged to decrease and/or quantify the resulting uncertainty in peptide-spectrum matches. Through PSM re-scoring, additional information (e.g., properties not related to the fragment spectrum or features orthogonal to the scoring process) is used to better discriminate between correct and incorrect identifications. This re-scoring can additionally be done in a way that the resulting new scores represent a probability of a *single PSM* belonging to either of the two classes, thereby quantifying uncertainty locally.

A different method tries to collect auxiliary information about the incorrect class and to estimate the rate of false discoveries among a *set of PSMs*. This is done by introducing so-called “decoy” peptides which are known to be absent from an experiment but otherwise resemble the actually considered (“target”) peptides in many features. Their scores can be used to guide the aforementioned re-scoring methods in a semi-supervised learning scenario, while the number of decoys in a resulting set of potentially identified PSMs helps in estimating the number of errors made among targets.

PSM re-scoring

In addition to the raw matching score from a database search engine, differences in peptide matching quality may be judged by many possible features, which can be roughly grouped into three categories: First, it is possible to score peptide matches based on the similarity between the predicted retention time of the peptide and the true retention time of the peptide[65,73]. Second, the proximity of the theoretical peptide mass to the measured precursor (i.e., in-tact, unfragmented) mass provides additional information. Third, orthogonal scores from the peptide spectrum matching process can be used to help in distinction. Such scores can be scores from other search engines with different scoring schemes[101], or an aggregate of information about the distribution of scores for every peptide candidate that was evaluated for a certain spectrum (hyperscores[28], e-values[72], relative score difference between top matches[40]).

Together, these statistics provide a rich collection of information about the probability of peptide identifications being correct or not. The qualitative statistical reasoning behind this is that one feature may match well just by chance while an increasing number of independent features raises the likelihood of a correct match.

It is important to note that the discriminative capability of the different features depends on many factors, such as the chosen database or the used mass spectrometer. With higher accuracy mass estimates from FT-MS[21] or Orbitrap instruments[58], it is possible to distinguish nearly identical masses (of precursors and/or fragments)[34] and thereby increasing the discriminative capability of features depending on mass accuracy.

Instead of empirically weighing those features, state-of-the-art techniques are learning weights or parameters from the data. Using the fact that it is a binary classification problem and that the features have a direction in which the likelihood of belonging to either of the classes steadily increases/decreases, allows an application of simple, unsupervised approaches[71,149]. Those try to fit two distributions (one for "bad" scores and one for "good" scores) to the mixed distribution (with hidden/unknown mixing proportions and parameters) of each feature (or a linear combination of a few). The posterior probability of belonging to the distribution of correct identifications given the values of its features can then be used for quantifying uncertainty of a specific identification under the given experimental and analytical settings. Although this approach was extended to use decoy information for the negative class[19], partial knowledge about class labels also allows the usage of other machine learning algorithms. The Percolator algorithm[67] for example uses a linear support vector machine to find the best separating hyperplane in the joint multidimensional feature space. Since labels have to be specified *a priori* for this algorithm, the tool uses decoy identifications to model the class of incorrect PSMs, while top scoring PSMs provide data for the correct class in an iterative manner. A logit transformation of the distance to the separating hyperplane for each PSM may then be interpreted as a (pseudo) posterior probability for belonging to a certain class. Its hyperparameters can be optimized with nested cross-validation. The PSMs can be sorted by the resulting posterior probabilities and the top n identifications used as a *maximum a posteriori* estimate for the subset of correct identifications when $\leq n$ PSMs are allowed to be true.

2.4.3 Quantifying uncertainty with the FDR

The goal behind target-decoy approaches is to estimate a statistical measure that quantifies the error in a (sub-)set of hypotheses, called the false discovery rate (false discovery rate (FDR)).

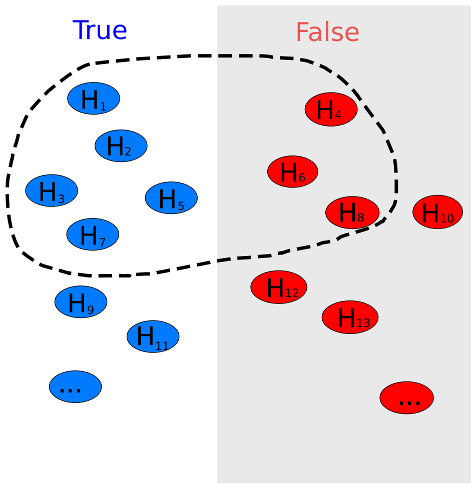
The FDR exists and is well-defined

The false discovery rate shows the proportion of false positives among all significant hypotheses, i.e., the percent of PSMs that were considered as reliably identified that are incorrect[9]. It is one of the most popular error measures for multiple hypothesis testing because it has an easily understood interpretation, is less conservative than the family-wise error rate (which controls having made a single false hypothesis) computed by the Bonferroni correction[35], and because the FDR can be viewed through a Bayesian lens for a specific hypothesis via the local "q-value"[151] (the minimum FDR at which a hypothesis would be considered as significant).

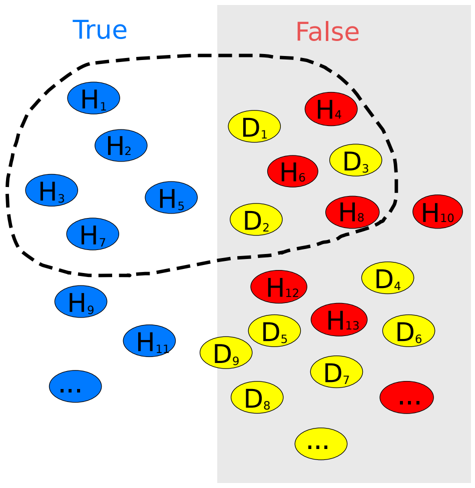
If every PSM is viewed as a hypothesis for a correct identification, some of them will be true and others will be false. As shown in Figure 2.5, a collection of hypotheses are considered for a particular problem, and a subset of that collection is selected as "significant", e.g., showing high enough matching scores. Then it can be seen that the FDR of this example is the proportion of the false hypotheses (H_4, H_6 and H_8) among the eight significant hypotheses. Thus, by specifying that we believe that the state (i.e., true or false) of those hypotheses has been fixed in advance, there should exist only one exact FDR for each considered problem.

The target-decoy FDR estimate

Unfortunately, the state (i.e., true or false) of the hypotheses is generally unknown, and so an important question arises: How can we compute the FDR in practice without knowing which hypotheses are true and which hypotheses are false? One approach is



(a) The FDR as a cardinality, a well-defined mathematical quantity.



(b) The target-decoy FDR estimate, one traditional estimate of the FDR.

Figure 2.5: The FDR and its target-decoy estimate. (a) For an arbitrary problem, a selected set of hypotheses (inside the dashed circle, which may have been identified or selected as "significant" by some previously executed procedure) is considered and evaluated. Target hypotheses are the actual questions of interest (labeled $H_1, H_2, \ldots$), whereas decoy hypotheses are questions that represent negative controls (labeled $D_1, D_2, \ldots$). Target hypotheses can be partitioned into two categories (which, in practice, are usually unknown): here, true target hypotheses are labeled in blue (H_1, H_2, H_3, H_5, H_7) while false target hypotheses are labeled in red (H_4, H_6, H_8). The exact FDR of the selected hypotheses is the proportion of false hypotheses among all the eight significant target hypotheses. **(b)** When the state of the target hypotheses is unknown, the decoy hypotheses can be used as an approximation for the behavior of false targets, and used to estimate the FDR (i.e., the target-decoy estimate): When the rate of decoy identifications is roughly the same as the rate of false positive target identifications (as is shown here), the number of decoys is an estimate of the number of false target identifications. Of the eight selected target hypotheses, three are estimated to be false positives (because three decoys were selected), yielding an FDR estimate of $\frac{3}{8}$, which in this example, is equal to the true FDR.

to rely on known, external evidence to infer an estimate of the FDR. Just as hunters set decoy animals to mimic their prey, the false positives can also be captured by using decoy hypotheses, which are known to be false and thus mimic the unknown false hypotheses and their score distribution in our problem. In this method, commonly called the target-decoy approach or strategy[97], the term target refers to a potentially true hypothesis of interest and decoy refers to these artificial, false hypotheses that are included solely for the purpose of statistical evaluation. If we assume that the chance for a false identification to be a false-positive (i.e., identified but absent) target peptide is roughly the same as being a false-positive decoy hit (Equal Chance Assumption), we can estimate the FDR[37] as follows:

$$FDR = \frac{\text{false target identifications}}{\text{total target identifications}} \approx \frac{\text{decoy identifications}}{\text{total target identifications}}.$$

This idea is shown in Figure 2.5.

Decoy hypotheses can be generated in many ways but ideally they must resemble the false target hypotheses and must also be necessarily false (thus they cannot overlap with any true target hypotheses). For peptide database search a few methods became established that allow robust estimates. For example, a decoy peptide database can be generated by randomizing, reversing or shuffling the digested target peptide sequences or by reversing or shuffling the target protein database and then digesting those reversed or shuffled proteins *in silico*[69]. Others proposed using databases from other species as decoys while considering the evolutionary distance between the decoy species and the target species of interest[166].

Regarding the estimation procedure itself, there are also multiple strategies: Some work suggests to search target and decoy databases separately[66], while others suggest performing both searches simultaneously by searching a concatenated target-decoy database[37]. The latter case is sometimes called “target-decoy competition”, because the search process produces competition between target peptide candidates and decoy peptide candidates for every spectrum searched. Therefore, the reported identifications contain a mixture of target and decoy PSMs which allows estimating the number of false positives by twice the number of decoys. A subsequent protocol for the target-decoy FDR estimate was proposed by Keich, Kertesz-Farkas, and Noble, who considered estimating false positives coming from native (generated by a peptide present in the target database) and foreign (e.g., decoy peptides or peptides with unexpected post-translational modifications) spectra separately using a mix-max procedure.[69]

Lastly, FDR estimation can be performed on different levels, e.g., spectrum level, peptide level, and protein level. To avoid inflation of the estimated FDR due to aggregational biases (i.e., from multiple PSMs per peptide or multiple peptides per protein) the estimation should be (re-)done on the level used to make conclusions about the data and not be carried over from other levels (without significant corrections, as for example described by Reiter et al. [125]). On a similar note, targets and decoys from multiple replicates or even samples should be merged before estimation to avoid hidden accumulation of false positives from multiple testing, when conclusions about aggregates of samples (e.g., whole tissues) are made[142].

2.4.4 Evaluating calibration of scores and FDR estimates

With target-decoy approaches one might be tempted to prefer scoring procedures solely based on their ability to separate target from decoy hypotheses. Besides the fact that targets consist of a mixture of correct and incorrect hypotheses, target-decoy methods are imperfect estimation procedures with potential biases and their power of estimating the null distribution is limited to the number of sampled decoys. Therefore, calibration should be performed on multiple levels to avoid wrong or biased estimates which are hard to detect without a ground truth.

Firstly, the input scores to target-decoy and re-scoring methods should be calibrated in a way that the meaning of a good score is independent of attributes of the tested hypothesis (e.g., the number of peaks in the spectrum, the precursor charge or the number of peptides tested against this spectrum). Here, miscalibration can be reduced by calculating p- or E-values for a PSM based on the score distribution from candidates for the same spectrum (e.g., via dynamic programming[72,74]), which ensures that the assumptions for most confidence estimation methods are met. Input scores can also be re-calibrated with a computationally expensive bootstrapping procedure[70].

Secondly, calibration can be measured for combinations of (re-)scoring functions and target-decoy methods depending on how well they approximate the true FDR at different quantiles. Besides simulations and ground truth samples of known content, this can be estimated by including a set of unlabeled decoys (often called entrapment decoys) which would reveal biases/overfitting to the current known set of decoys.

Lastly, if inference is done probabilistically (i.e., if the scores that are reported are well-defined (PPs) or (PEPs)), then the FDR can be seen through a Bayesian lens, and can thus be estimated from the probabilities. Since probabilities are a local measure of a specific hypothesis being true, the FDR can be estimated by the expected proportion of false positives in a set. If the hypotheses are roughly independent, this is computed as the sum of posterior error probabilities divided by the cardinality of the set. For example, if H_1 has posterior probability 0.7 and H_2 has a posterior probability of 0.1, then identifying H_1 yields an expected fractional 0.3 false positives and identifying H_2 yields an expected fractional 0.9 false positives, meaning that in total 1.2 false positives are expected. This yields an FDR of $\frac{1.2}{2} = 0.6$. The calibration can then be thought of as the similarity between this probability-based false positive estimate and the ground truth false positive count. Probabilistic methods therefore provide complementary means to estimate the FDR (in place of target-decoy search).

2.4.5 Protein inference

Certain technical and analytical obstacles in the identification of intact proteins[23,162] made pre-MS protein digestion a common approach in proteomics experiments. An important side effect of digestion is the introduction of an additional step to the analytical process in the common case where researchers are interested in conclusions about the composition of intact proteins in the investigated sample: a step to map the putatively identified peptides back to the most plausible set of source proteins, the protein inference problem[107,145,163]. Aforementioned, relevant experimental and analytical steps leading to the protein inference problem are depicted in Figure 2.6. The constructed example in Figure 2.6 also shows analytical challenges and ambiguities that may arise in shotgun proteomics experiment.

More details and an important representation of the problem are described in the following part of this introductory subsection which was moved from the manuscript in Chapter 4 to provide early background information (see that Chapter for the contribution table):

EPIFANY - A method for efficient high-confidence protein inference

J. Pfeuffer, T. Sachsenberg, T. M. Dijkstra, O. Serang, K. Reinert, and O. Kohlbacher

Journal of Proteome Research p. 734327, (2019), Copyright 2019 American Chemical Society.

High dynamic range of protein abundance, limitations in digestion, separation, and mass spectrometry result in incomplete coverage of the source proteins by identifiable peptides. Reconstructing the source proteins originally present in the sample should thus rely on as much of the experimental evidence (i.e., peptide identifications) as possible. This includes degenerate, non-unique peptides shared between multiple source proteins. Starting from scores or probabilities ranking peptides by their likelihood of presence, we want to infer the presence or absence of the proteins these peptides originated from. Due to the common presence of ambiguous peptides arising from one or more proteins sharing parts of their amino acid sequence this is not a trivial task.[107]

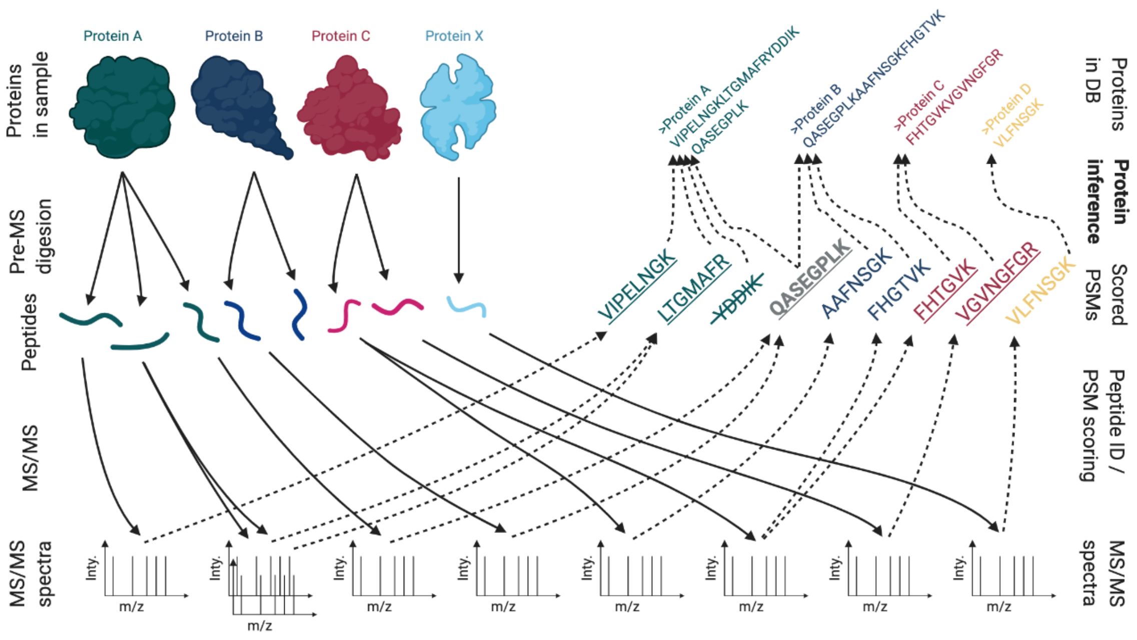


Figure 2.6: Experimental (left; solid arrows) and analytical (right; dashed arrows) steps leading to the protein inference problem (top right; bold) in shotgun experiments. After enzymatic digestion of the proteins, if enough copies of a peptide are ionized, it is selected for MS/MS and one or more fragment spectra are generated from it. In the first analytical step, they are mapped to peptides from a theoretical digest of a candidate protein database and are scored (yielding scored PSMs). Note, that due to an incomplete database or imperfect scoring, peptides might be identified incorrectly (not underlined). If two proteins share a theoretical peptide and this peptide is matched to an experimental spectrum, ambiguities arise (bold gray). Some theoretical peptides (here marked as scored out) might not be matched at all, e.g., due to incomplete digest or insufficient ionization. Protein inference uses scored PSMs and their associations to parent proteins, to rank and score the proteins according to their likelihood of presence.

The formulation of protein inference algorithms naturally leads to a representation of the relation between peptides and proteins as a bipartite graph of nodes (proteins and peptides) that are connected with an edge if a peptide is part of the theoretical digest of the (parent) protein (Figure 2.7). Figure 2.7 also shows that the ambiguity of peptides across proteins may lead to proteins without unique evidence (e.g., protein E) and in an extreme case to experimentally indistinguishable protein groups (e.g., one comprising proteins F and G, which share all their observed peptides). Reasons for ambiguous peptides are manifold in biology and include among others homology, alternative splicing or somatic recombination. Depending on the degree of ambiguity between the peptides of different proteins they are often clustered into various types of so-called protein ambiguity groups.[107] It should be noted that those groups can either be defined solely based on the experimentally observed peptides or based on all theoretically possible peptides.[147] Preliminary grouping (especially of indistinguishable proteins) is often used automatically by inference algorithms to solve a less ambiguous problem. While this is enough for studies that only need to confirm presence of any protein in a group (using group level inference probabilities), other studies look for the effects of a certain isoform on a disease and need to know how likely it is that this specific isoform was expressed (therefore interested in single protein level results). To compare results of inference methods on the same level (wherever possible) is an important consideration during benchmarking.

As of today, a multitude of methods exist to tackle the protein inference problem, which are categorized and shortly explained in the following paragraphs.

Rule-based methods

Because of their simplicity, rule-based methods were among the first methods to be used to solve the protein inference problem[16]. Their outcome is pure binary. When a specific value or quantity associated with a protein exceeds a certain threshold it is said to be present, otherwise it is reported absent. Rule-based methods come in different flavors. They mainly differ in the variable on which the rule(s) are applied. The simplest form only looks on the number of identified peptides (independent of how this is decided) that may stem from the digestion of the protein to evaluate. Although this group of methods is easy to implement, has transparent criteria and is fast in execution, it faces one particular problem: Thresholding all proteins according to the same rule, possibly over different datasets is an overly simplified view on the actual

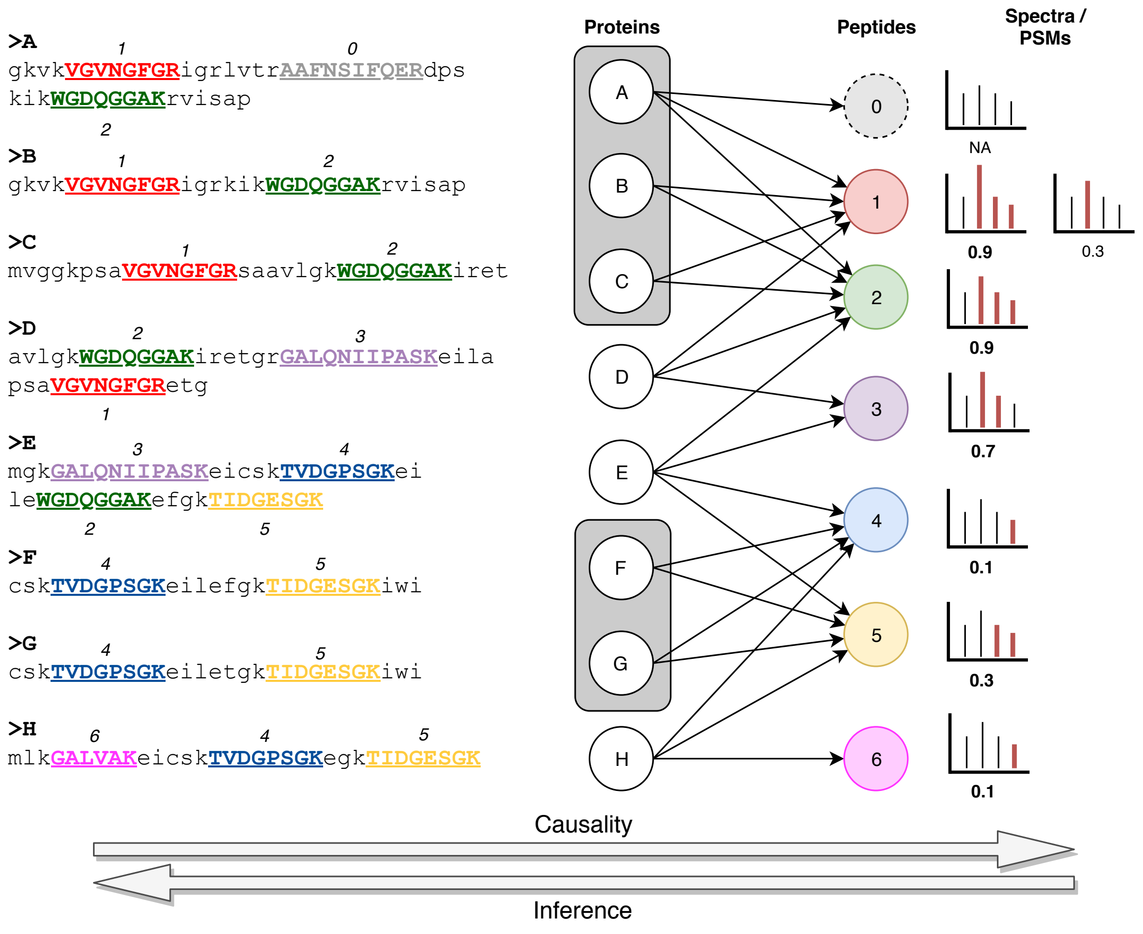

>A
gkvkVGVNGFGRigrlvtrAAFNSIFQERdps
kikWGDQGGAKrvisap
>B
gkvkVGVNGFGRigrkikWGDQGGAKrvisap
>C
mvggkpsaVGVNGFGRsaavlgkWGDQGGAKiret
>D
avlgkWGDQGGAKiretgrGALQNIIPASKeila
psaVGVNGFGRetg
>E
mgkGALQNIIPASKeicskTVDGPSGKei
leWGDQGGAKefgkTIDGESGK
>F
cskTVDGPSGKeilefgkTIDGESGKiwi
>G
cskTVDGPSGKeiletgkTIDGESGKiwi
>H
mlkGALVAKeicskTVDGPSGKegkTIDGESGK
Proteins
Peptides
Spectra / PSMs
NA
0.9
0.3
0.9
0.7
0.1
0.3
0.1
Causality
Inference

Figure 2.7: Example of a bipartite protein-peptide graph. Nodes with letters represent potential proteins from the input database. Colored nodes are the peptides from *in silico* digest with the given enzyme (trypsin). Arrows are drawn when a protein may theoretically generate a peptide. Dashed circles represent experimentally unobserved entities due to missing ("NA") peptide-spectrum matches. Red peaks in the sketched hypothetical tandem mass spectra were matched to a theoretical ion of the peptide that matched best to this spectrum. Probability scores roughly follow a dot-product based score but were invented for the sake of this example. Bold scores highlight the chosen match probability for this peptide (i.e., the maximum probability). The left side shows the used protein database with their tryptic peptides (upper-case bold underlined substrings) following the same color and number scheme as the nodes in the graph. Proteins in the same shaded curved rectangle comprise an experimentally indistinguishable (ambiguity) group. The arrows on the bottom show the general directions of the two processes: causality in the course of the experiment and inference based on the observed data.
This figure and its in-text description were part of the publication in Chapter 4. They were moved to provide early background information.

process and prone to overcount evidence from shared peptides. In a sample with a large protein, producing many peptides (e.g., titin), if we accepted every protein that is connected to one of the identified peptides, we would count the peptides that are already explained by the presence of titin multiple times and falsely identify a lot of hitchhiking proteins[124]. Especially if the protein does not have other unique peptides (i.e., "one-hit wonders" or subset proteins in general). To avoid this, one could use the two-peptide-rule and require that a protein has at least two identified peptides as neighbors. This will reduce the amount of false identifications that come from sharing peptides with titin—though not necessarily eliminate all of them. However, it also prevents correct identification of (primarily short) proteins for which only one of its peptides was found, therefore making false negative decisions[52]. One can see that one should not aim to find one general rule for every protein. For example, we know *a priori* that for a longer protein with more theoretical peptides, it is more likely to observe at least one peptide from them (compared to shorter ones). Also, the choice of the best rule will differ from dataset to dataset. For example, given a dataset, where not a single observed peptide is shared, a one-peptide-rule will probably yield the best results (dependent on the performance of the preceding peptide identification process). Although the uncertainty about the peptide-spectrum matching can be taken into account, for example by weighting the number of peptides observed for a given protein by their estimated probability of being correct, a better handling of shared evidence is necessary to approach the problem in a more rigorous way.

Parsimony and optimization-based approaches

One way to avoid overcounting evidence and accepting too many (supposedly false positive) subset proteins is to follow the parsimony principle – sometimes titled Occam's razor[33] – and pick the least complex of all hypothetical solutions.

In the context of protein inference, using the parsimony principle means picking the smallest set of proteins (in the input database) that explains all observed peptides. From a combinatorics perspective, computing the parsimonious set of proteins is equivalent to solving the minimal set cover problem by introducing one subset for every protein, consisting of the peptides it explains, and trying to cover the full set of observed peptides with as few subsets as possible. It is often relaxed to the related partial set-cover problem, where only a fixed percentage of the peptides have to be explained (since some peptides might have been falsely identified by noisy spectra). Both problems can then be formulated as an integer linear program (ILP). A general form is depicted in the following equation:

$$\begin{aligned} \text{minimize} \; & \sum_{i=1}^{m} r_i \\ \text{subject to} \; & \sum_{j:r_j \in \mathrm{NB}(p_k)} r_j \geq 1, \; 1 \leq k \leq n \\ & r_i \in \{0,1\} \qquad , \; 1 \leq i \leq m \end{aligned}$$

Here we look for the minimum number of proteins that have to be present ($r_i = 1$) such that at least one of the neighboring proteins ($x_j \in \mathrm{NB}(p_k)$) for every peptide p_k is present (e.g., sum greater than 1). Value m denotes the number of proteins and n denotes the number of peptides.

One issue with methods modeling the problem in this way is that algorithms solving them generally are in the same class as other very hard problems in computer science, the class of NP-complete problems, for which exact fast solutions are generally believed to be impossible[68]; however, the first methods acknowledging the parsimony principle[88,174] were able to approximate the result by using a greedy heuristic that iteratively picks the protein that covers most of the remaining peptides and adds them to the set of present proteins. This is repeated until every peptide is explained. Although the results of greedy methods can be shown not to be worse than a factor smaller than log(#peptides)[150], in comparison to the best possible solution, they often seem very unintuitive in the cases where this suboptimality occurs. Other approaches therefore try to relax the problem by allowing non-integer solutions (i.e., scores) for the proteins, thereby creating a (non-integer) linear program (LP) which can be solved much faster[103] than its binary version.

Linear and quadratic programming

Both greedy and LP methods have been used to compute solutions for the protein inference problem on peptide mass fingerprinting results and database search results or combinations thereof[54]. There are also programs that do not use the set-cover formulation explicitly but try to minimize a norm of the protein presence vector (i.e., reminiscent of regularization procedures) setting as many values for proteins to 0 as possible. Based on the ideas of Noble and Weston, this was performed using the Euclidean norm (also called L_2-norm), which then represents a quadratic program (QP) that could be solved efficiently with a method by Serang[138].

Shortly after that, others reformulated the problem as an optimization problem on the adjacency matrix of the bipartite graph. That means columns represent proteins, rows represent peptides. The formulation of the LP is then created in a way that it maximizes the number of zero-only columns (absent proteins) in that matrix such that the observed peptide probabilities are always close to be preserved as the sums of rows[59].

Generally, once an LP or QP is formulated, it is relatively easy to use it as input together with different input data, given that one of many user-friendly solvers using well-proven algorithms[102,103,138] is available.

Although the concept of parsimony seems to be reasonable to avoid false positive proteins, it might be overly restrictive to base the model on the assumption that each peptide was only generated by at most one protein. Once explained, these methods are basically forbidding other proteins that could generate the same peptide. As an example, it does not allow any coexistence of proteins that share exactly the same set or a subset of observed peptides ("indistinguishable proteins" or "subset proteins" as defined by Nesvizhskii and Aebersold[104]) and simply chooses a random representative of the proteins that cover most peptides. One solution is to additionally include unobserved peptides and peptide detectabilities[45,90,157] like in the minimum missed peptide formulation described by Alves et al.[4]. A more general approach however, is to accept the uncertainty about the association of proteins to peptides by modeling it in a probabilistic manner.

Probabilistically motivated heuristics

After recognizing the uncertainty inherent in the peptide identification process as well as in the generation process (from proteins to peptides), an answer about the presence of a protein should also be not a simple yes or no. This can be addressed by treating every protein and peptide as a random variable for which we can define and infer probability distributions, in the simplest case a single probability of being present. We are especially interested in the conditional probability of the presence of each protein given the observed evidence for the peptides or PSMs. This probability, after including evidence, is the so-called posterior probability. Again, there are many options to model the details of a probabilistic approach and an equally hard decision has to be made on the algorithm used to solve it.

ProteinProphet

One of the first probabilistically motivated inference methods that were developed was ProteinProphet[105] which is still widely used in the field of computational proteomics. It bridges the gap between methods based on the parsimony principle and fully probabilistic models. For the sake of lower computational complexity the authors make a simplifying assumption that they later correct through incorporating substitutional variables which are estimated in iterative, expectation-maximizing procedures.

The assumption that is made by ProteinProphet, is that peptides are generated completely independent of each other. This is however not true for peptides coming from the same protein: If we observed one of its peptides, it is now more probable that the other peptides of this protein are also present. This is then corrected by re-weighting the peptide probabilities according to the number of sibling peptides they have for a given protein, which is the sum of probabilities of all other peptides that come from this protein. The method uses this to increase the probability of peptides in multi-hit proteins and decrease the probability of peptides in single-hit proteins, which are assumed to be less likely to be correct identifications. This clustering of correct peptides to fewer correct proteins is also the reason for the increase of FDR from peptide level to protein level as discussed in Section 2.4.3.

Another undesirable effect of the independence assumption of peptides is that proteins are implicitly assumed not to affect each other despite having a shared peptide, which is not true in reality, where in such a case, the presence of one protein “explains away” the presence of another protein, essentially fighting about the evidence from the shared peptide. Again, acknowledging this flawed assumption, the authors incorporate a notion of sharedness of peptides by introducing weights for every protein-peptide association. These weights sum up to one for every peptide and are updated in every round of the algorithm to converge to a reasonable partitioning of this peptide’s evidence to the proteins it might stem from. This also guarantees that the evidence from a peptide is (in total) only taken into account once and therefore resembles parsimony-driven methods.

However, despite these elaborate ways to cope with the heuristic nature of the model and its usually good performance in practice, there are limitations to this approach. One of them is its lack of robustness in certain situations. Known examples are bad handling of subset proteins and a tendency to extreme probabilities of proteins that are close to be indistinguishable. Li and Radivojac[81] describe these situations in greater detail on an example. Other probabilistically motivated heuristics similar to ProteinProphet, such as EBP[121], try to overcome such limitations by including additional steps in the iterative estimation procedure (e.g., estimating the most likely protein from a homology group), increasing the number of parameters in some steps (e.g., estimating parameters for proteins of different abundances separately) and using more complex parameterized conditional distributions (e.g., Poisson distribution for correct matches between proteins and peptides); however, although the aforementioned undesirable effects of these issues might be reduced with certain settings, tweaks or postprocessing, it might still be favorable to get down to the root of the problem and remove overly simplistic assumptions (like fully independent peptides).

Generative models

Modeling these dependencies is possible with generative models. They inherently make use of the information from the bipartite graph (Figure 2.7) that we can create from our prior, expert knowledge about the generation process of the data while still incorporating uncertainty in the predictions. This is done by modeling every quantity involved in the process as a random variable and specifying dependencies between them. This can be visualized as a graph where nodes represent random variables and edges denote dependencies. By using directed edges we can additionally introduce a notion of causality between variables. These directed acyclic graphs (DAGs) are often called Bayesian networks (BNs). Bayesian networks are both mathematically rigorous and intuitively understandable. They permit an efficient, factorized representation of the joint probability distribution over the set of random variables given their known dependencies. They also allow structured computational inference procedures to be used that outperform brute-force enumeration[113]. The goal in protein inference is to infer posterior marginal probabilities for the proteins given the observed data, which is usually the likelihood of peptides being present (computed from PeptideProphet probabilities or similar, see Section 2.4.2).

Tree-based approximations

Still, there are different levels of detail one can use when constructing such a network. An improvement on the fully independent peptides assumption in many early models was made in the work of Li, MacCoss, and Stephens [80]. They correctly assumed that, if at all, the probabilities of peptides stemming from the same protein are only independent given that we know something about the presence/absence of their common parent protein (i.e., are conditionally independent). Because in that case, knowing that one peptide has a high probability of being present does not change anything in our beliefs on another peptide from the same protein. If the protein was absent, then they were identified from different proteins or noise. If it was present, then we know from the presence of the parent protein that both peptides were quite likely to occur, and knowing that one was present for sure, does not raise our beliefs (anymore) that other peptides are present. However, they still assume that proteins are independent given information about the peptides by introducing a separate copy of every peptide to every protein in the graph it came from, which makes the graph a tree and is again a simplification. Apart from improvements in the assumptions, they proposed parametric formulas for the number of peptides that are identified given information about protein length and presence, as well as for the PSM scores for peptides given their presence/absence (similar to PeptideProphet but in a combined fashion). The parameters are learned via an expectation-maximization (EM) algorithm, which then enables the computation of posteriors for proteins and peptides.

The hierarchical statistical model (HSM) by Shen et al. also uses an EM algorithm to learn parameters but introduces different conditional distributions and is among the few methods acknowledging the dependence of proteins in the same connected component through shared peptides[148]. They again take into account the number of peptides observed for a protein and model the scores on PSM level as a mixture of correct and incorrect distributions but also consider properties of protein cleavage and introduce another layer to allow for misassignment of PSMs.

General graphical models

However, there are even more general models of this class (e.g., the ones from Fido[135], MSBayesPro[82,83]) that use minimal assumptions regarding shape of the conditional distributions. Complicated, highly parameterized distribution shapes (e.g., distributions with multiple modes of different heights) are often well justified by empirical research but lacking extensive theoretical justification.

Other than the dependencies implied by the bipartite graph, these Bayesian networks need a parameter to be set for the prior probability of the proteins (like the HSM does). The conditional distributions of the peptides given their associated proteins are modeled in way that can be seen as a form of the probabilistic- or noisy-"OR" function. A peptide can only be present if at least one of its parent proteins is present but there is only a certain probability α for every edge (e.g., protein-peptide combination) with which a present protein "triggers" the presence of the peptide (i.e., generates that peptide). A good approximation that MSBayesPro uses for this generation probability is the peptide detectability of the peptide in a protein; however, estimating this detectability from empirical data is another machine learning problem[45,90,133,167] that relies on certain assumptions and requires additional runtime. Fido makes a simpler approximation, using one fixed value for all combinations and adds a “leak” node that allows a peptide to be generated by noise (with an own probability β). Both methods are able to make use of unobserved peptides to facilitate the distinction between proteins with the same set of observed peptides. The authors also took different paths when it gets to inference on these models. While Fido tries exact brute force enumeration and resorts to pruning the graph only when the algorithm becomes infeasible, MSBayes relies on a stochastic method called Gibbs sampling where the degree of approximation is inversely proportional to the number of iterations it ran.

A similar approach called MIPGEM[46] differs from the aforementioned two by modeling the conditional distribution of peptides given all proteins are absent as uniform in the range of all observed (logit-transformed) peptide probabilities, given that at least one protein is present as a piecewise linear density with one free parameter. However, to speed up inference, the authors added a Markovian assumption where only neighboring proteins are thought to affect the conditional distributions of a peptide (which again limits the ability to “fight about pieces of evidence” among a chain of proteins). Besides that, inference is done by brute-force enumeration but falls back to sampling, when the number of possible proteins sharing a peptide exceeds ten. All three methods make use of the fact that different connected components in the graph can be solved independently.

On a gold standard dataset it was shown by The et al. [160] that fully probabilistic models perform among the best in terms of the pure identification task but in general, the fact that these models work on discrete probability tables, results in a combinatorial explosion of possible explanations for a peptide, in cases where it is shared by many proteins. This is the reason for pruning and sampling currently being the last resort for inference in Bayesian networks on complex eukaryotic samples.

Intractability of inference in Bayesian networks

Although representing the protein inference problem as a Bayesian network became a successful approach to tackle the protein inference problem, it is still far from being a perfect solution. The generalizability of Bayesian methods is probably their biggest weakness as well as their biggest strength at the same time. This certainly has to do with the rapidly increasing complexity of inference for every additional detail introduced. With brute force enumeration, even the simplest models (still recognizing all dependencies in the bipartite graph) are hard to solve. This is probably one of the reasons for the recent lack of emergence of more detailed (Bayesian) models for the protein inference problem.

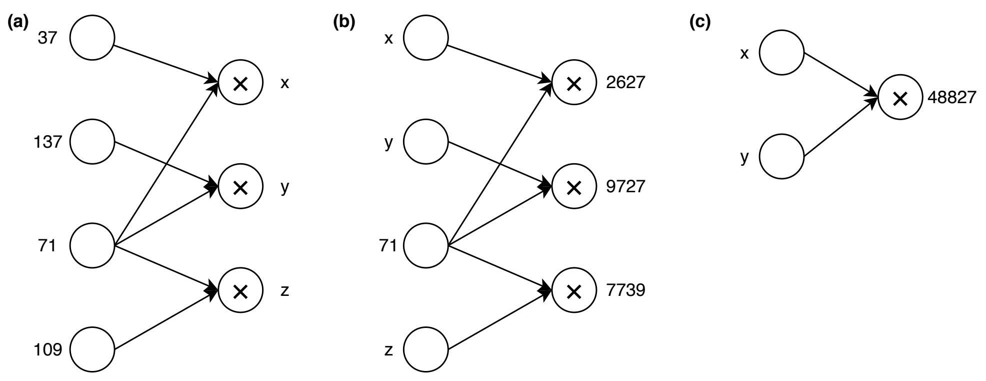


Figure 2.8: Intuitive reason why converging dependencies (i.e., multiple arrows going into a node) are hard. Blank nodes (left layers) represent input factors, the nodes with a multiplication sign "×" (right layers) visualize the resulting product. Numbers indicate a known value, whereas variables indicate an unknown value. **(a)** In easy problems, where all factors are given, we can simply proceed in a left-to-right manner with simple multiplication (e.g., $z = 71 \times 109 = 7739$). **(b)** In nearly identical problems, the products are given and a single factor is divided out compute the results in a right-to-left manner. **(c)** In these extremely difficult problems, the product $x \times y$ is given, and we are tasked with finding integers x and y that achieve that product. For these problems (even when only two variables are considered, as shown) there is no known general solution that is substantially more efficient than trying all possible pairs x, y, which becomes roughly exponentially difficult in the number of digits given; however, such problems (which are said to be in the complexity class "NP") can be easily verified given the correct solution $x = 157, y = 311$, we can easily verify that $x \times y = 48827$.

In graphical models, special computational difficulties arise when we encounter a large set of converging connections (i.e., many nodes pointing onto one, as shown in Figure 2.8), which converge to produce a known result. Such graphical problems can easily be seen to be at least as difficult as prime factorization used in many modern encryption algorithms[127]). Thus, we can see that when arrows converge, and we are expected to perform inference right-to-left (i.e., in the opposite direction to the causal flow of information), the problems can easily be quite difficult.

Intuitively the task of picking the most likely set of proteins that generates a set of peptides with observed probabilities is of similar difficulty, with the clusters of converging edges corresponding to a single shared peptide and its neighboring proteins. It could even be fully proven that Bayesian inference in general is NP-hard[24] by showing that it can be used to decide on the satisfiability of a quite general form of logical formulas (3-SAT problem).

Current solutions for protein inference on discrete Bayesian networks

Although there are already methods that nonetheless allow us to draw conclusions from Bayesian networks based on the bipartite graph, they either add simplifying assumptions (e.g., graph pruning in Fido) or may take arbitrarily long to reach a reasonable result close to the exact solution (Gibbs sampling in MSBayesPro). One upcoming family of approaches with the goal of reducing the computational complexity of graphical models are “variational methods”, which attempt to model the ideal posterior distributions as the optimum (i.e., the distributions which achieve the minimum free energy) for a given graphical model and corresponding data[61]; however, the benefits of these approaches are rather problem-specific.

A more general strategy to circumvent brute-force enumeration of all states is to apply a tree-decomposition on the Bayesian network to generate a so-called clique or junction tree[146] to avoid redundant computations. On trees, efficient message passing algorithms[62,78,112] exist that can calculate exact posteriors for all variables with two linear passes from a root to the leaves (and back). A smart tree-decomposition basically tries to combine random variables into higher-dimensional clique nodes and to balance the cost of inference on the graph as a whole versus the complexity of marginalization inside each of the combined clique nodes.

Tree-decomposition

Junction trees can be constructed by applying the following steps on a Bayesian network: moralization (connecting parents of a shared child and removing edge directions), triangulation (to ensure that if a variable appears in two cliques, it appears in each node on the path between the cliques; the running intersection property) and clique finding. The complexity of inference on the resulting junction tree with standard probability table representations is then exponential in its treewidth (i.e., the cardinality of the largest resulting clique). Finding an optimal tree-decomposition with a triangulation that produces the smallest maximal clique is NP-hard on its own[5], such that one usually resorts to heuristics like min-weight, min-fill or max. cardinality search (reviewed in Musliu[100]). However, applied to the Bayesian network of the protein inference problem, subgraphs corresponding to one degenerate peptide and all of its proteins will always form a clique after moralization (Figure 2.9). With standard table-based representations (e.g., named tensors) of the involved random variables, the storage of their probabilities (or in unnormalized form, the so-called *potentials*) and the number of computations for updating and marginalization are therefore exponential in the size of the *domain* of the clique (i.e., the number of random variables on the clique, in our case the number of proteins sharing a peptide plus one). The variables shared between neighboring cliques about which the cliques need to exchange information in a belief propagation algorithm are often called *separators* and also play a role in complexity considerations during belief propagation.

Belief propagation algorithm(s)

Belief propagation on junction trees can be performed with different architectures[78]. We will sketch the HUGIN algorithm[62] in the following. Other architectures have the same core idea but make trade-offs between runtime and memory usage by caching/storing less or different potentials (e.g., current temporary potentials versus initial potentials only). The HUGIN algorithm for example requires storing current potentials both on cliques and on separators (essentially forming a hypergraph). At first, each initial prior or conditional probability table is associated to the smallest clique containing all of table's variables in the clique's domain. Then, cliques are initialized with the product of the potentials of all variables associated to it. The initial potentials of the probability tables are defined through the parameterization of the Bayesian network. If evidence is present for a variable, it has to be incorporated in the initial potential as well. Products of potentials are defined by named tensor multiplication (i.e., tensor multiplication of the dimensions of the intersecting variables). Potentials on separators are initialized uniformly with ones (the identity potential). To start the algorithm, we choose an arbitrary node as a root. Each node in the tree then follows the following rules:

- We say that a node is ready to send a message to a certain neighbor, if it received messages from all its *other* neighbors. This implies that propagation starts in the leaves.
- To make sure the propagation needs two passes only, the root holds back sending messages to its neighbors until it has received messages from all of them.
- Once a node is considered ready for a chosen neighbor (see first item), it calculates the message by marginalizing its current potential to the intersecting subset of variables with this neighbor. It then proceeds by sending the message to the separator on the edge between it and the neighbor. Marginalization of potentials stored as tensors to a subset of variables is done by performing a summation or maximization for every state of the subset domain over all states of the whole domain that are compatible with the subset state.
- As soon as a separator receives a new message from any of the two nodes it connects, it has to divide the message by its current potential and passes the result to the opposite node. After that, it substitutes its now outdated potential

with the received message. Division of potentials is analogous to multiplication with the exception that for any pair of values to be divided 0/0 is defined as 0.

- Nodes which receive a message, multiply their current potential and the incoming message and subsequently replace their old potential with the resulting product. Re-normalization is advised to prevent numerical underflow.

After the two passes, each clique and each separator holds the joint posterior probability of all variables included in it. Therefore, to query the posterior of a specific single variable, one needs to find the separator or clique with the fewest variables containing it and perform a last marginalization if the node is multidimensional (i.e., more than just the requested variable is represented by that node). For rigorous mathematical formulations of each rule please see for example Lepar and Shenoy[78].

Reducing complexity with convolution trees

As we saw, the largest improvements are to be achieved by reducing complexity on large cliques where storage and marginalization is very costly. Breaking up complexity in large cliques in general requires recognizing symmetry in the joint probability distributions for each clique employed. One such method exploits previously unrecognized symmetry in cliques from shared peptides[141]. As long as the conditional probability for a peptide is dependent on an additive function of the presence (or quantity) of its neighboring proteins— as in the noisy-“OR” model of the tool Fido with fixed or protein dependent parameters— inference can be sped up by performing fast Fourier transformation (FFT)-based convolutions along a binary tree. The method generalizes to local (discrete) conditional probability distributions of arbitrary dimensionality whenever the output depends on additive or cardinal operators. This approach can be used with both binary and quantitative distributions (as long as the discrete distributions are regularly spaced, which is required to enable fast convolution via FFT). This reduces the runtime on a single subproblem significantly, since only a small number (logarithmic in the number of proteins) of fast convolutions have to be performed. The number of operations then reduces to “$O(k\log(k)^2)$ and the space to $O(k\log(k))$ where k is the number of variables joined by an additive or cardinal operator”[141] (here, the number of proteins).

Partial incompatibility of junction tree inference and convolution trees

Ideally, one would try to combine convolution trees with aforementioned, usually faster junction tree inference to avoid expensive marginalization in potentially large resulting cliques. However, there is the following problem: dependent on the network structure, the required tree decomposition may create cliques that overlap in many variables in the case of multiple shared peptides. During belief propagation this leads to high-dimensional messages about the state of these variables for these separator potentials. These messages need an exponential amount of space to be stored and their incorporation/passing is equally complex. An example for such a case can be seen in Figure 2.9. Additionally, although convolution trees can be used to efficiently represent the internal structure of cliques when all variables in the clique are solely connected due to an additive dependency, it is currently unclear if there is an efficient tree decomposition algorithm with reasonable resulting treewidth that can guarantee this internal structure for every clique even in more complex networks.

Loopy belief propagation

The missing link between solving those hard subproblems on cliques generated by a shared peptide with convolution trees and propagation of their information for general inference on the full graph is a method that allows the exchange of information between those cliques on a lower-dimensional level (preferably on the single protein level). Since proteins commonly are part of several such converging patterns and their inferred probabilities in each of them do not necessarily agree, an iterative method is needed here. In other words, such a method needs to handle the cyclic dependencies that remain in the graph because of waiving the completeness of the tree decomposition and its running intersection property. A method that is capable of doing this, which shows good results in similar hierarchical “pyramid” networks[2,98] and became famous in message decoding[94] is loopy belief propagation. The method does not require the—in general NP-complete—creation of junction trees and may be applied on a slightly modified version of the original Bayesian network (the factor graph, see Chapter 4). Under the reasonable assumption[98] that in a problem of difficulty n, at most n iterations of loopy belief propagation are needed for the results of different clusters (in a connected component) to converge, we show on the bipartite graph of the lung data from one of the human proteome drafts by Wilhelm et al. that this leads to a compelling shift to lower complexity problems (Figure 4.6)[170]. Additionally, this inference procedure combined with convolution trees allows to explore the advantages of a second type of inference (max-product inference[136]), if the numerical stability of this method can be improved (see Chapter 3). Max-product inference would be less prone to produce smoothed posterior probabilities due to the central limit theorem, with only minor drawbacks in runtime and accuracy.

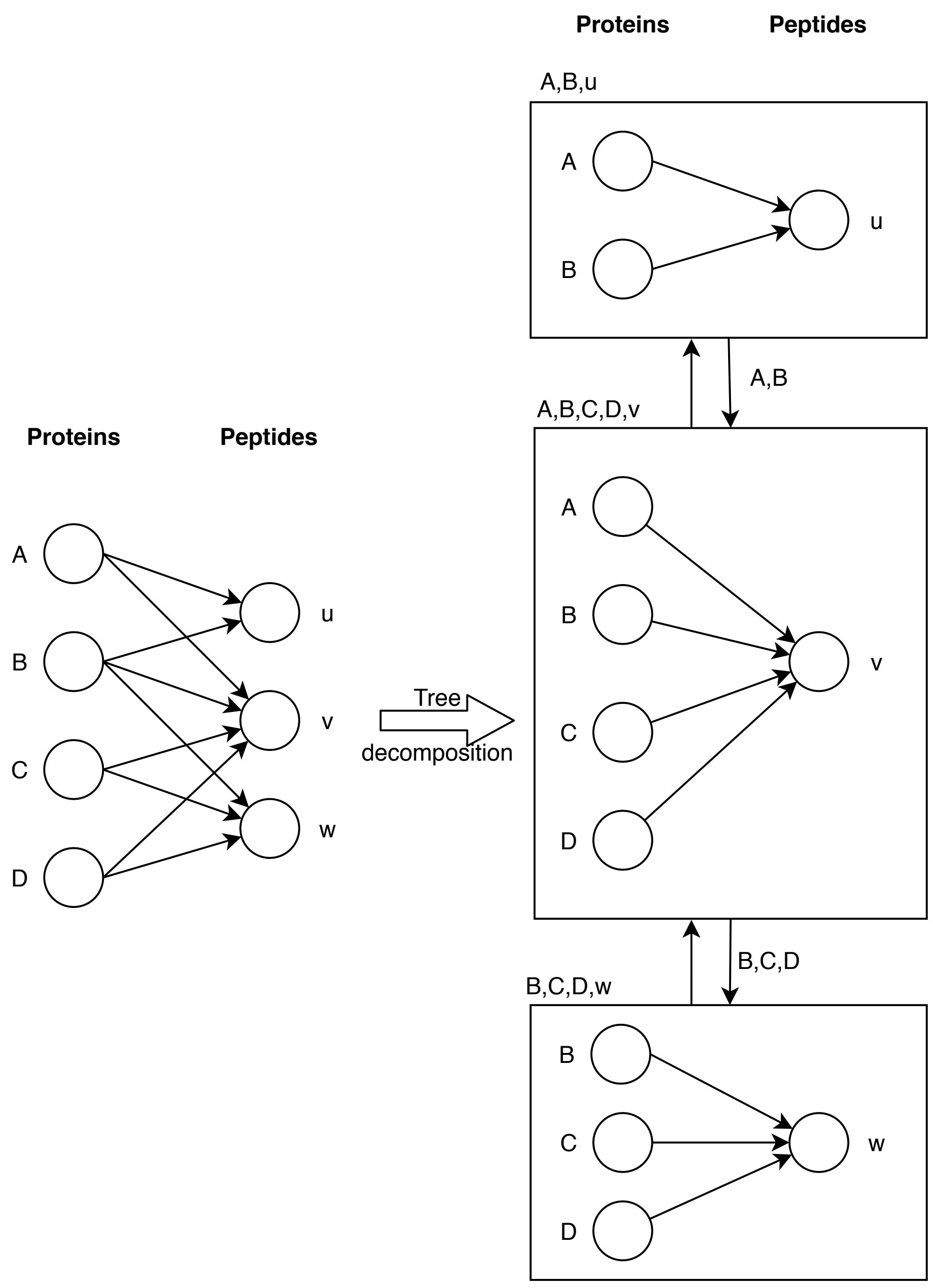


Figure 2.9: Tree decomposition and junction tree inference with message-passing Tree decomposition for the left Bayesian network yields the junction tree on the right. Boxes denote cliques with their domain variables written on their top-left corner. Applying belief propagation on the tree (e.g., with the HUGIN algorithm), requires passing of messages of sizes equal to the product of the cardinalities of each variable on a clique-connecting edge (so-called separators). When all variables inside a clique stem from an additive dependency structure in the original network, convolution trees can be used to reduce the complexity of marginalization and memory consumption of their representation inside the cliques.

2.4.6 The OpenMS framework

To be able to analyze a shotgun proteomics experiment from the raw spectra to quantities and identification probabilities of proteins usually means choosing a set of specific tools for the various steps mentioned above. Due to the high variability of instruments, experimental setups, research questions and available hardware it is beneficial to resort to frameworks that bundle all tools for a complete workflow in an integrative environment to ensure compatibility and ease-of-use.

One such framework is the BSD3-licensed mass spectrometry software package OpenMS. In addition to the provided tools it also allows for easy modification of the existing functionality and development of efficient new algorithms through its underlying C++ library.

The OpenMS core library

The core library is based upon flexible data structures for spectra, maps, PSMs, peptides, and proteins. Additionally, it provides algorithms and functions ranging from basic math and statistics to more advanced applied algorithms that build the basis of its tools. A core part comprises the efficient handling of open mass spectrometry file formats that empowers the integration with community tools outside the framework. It makes use of many features of the C++ 17 standard and the associated algorithms in the C++ standard library plus some extensions from Boost and Qt 5 (e.g., for user interfaces). Its modular CMake-based build system allows the inclusion of other externally developed libraries, for example if provided and maintained by a third party or if preferred to be distributed as a separate module. The source code is hosted publicly on GitHub and is well-tested for code style, coverage and functionality through automated continuous integration combined with manual code reviews.

The OpenMS tools

A full OpenMS distribution also ships over 180 ready-made tools, including a complete set of command line utilities to perform protein identification and quantification of a shotgun proteomics experiment starting from the raw spectra (as outlined in previous sections). While most of the tools are based solely on the OpenMS library and can be executed standalone, others require bundled third-party software (such as raw-file converters, Percolator or spectrum search engines). Apart from that, the toolset allows for basic signal processing of mass spectra, file conversion, and the analysis of other types of mass spectrometry data e.g., from SWATH, metabolomics, cross-linking or top-down proteomics experiments. The tools are available through installers for all major operating systems, bioconda[50] and the Dockerhub container registry.

Integration into Workflow Systems

With the existence of this large variety of tools of targeted functionality, a platform to easily chain those building blocks into comprehensive workflows is essential. OpenMS provides means of integration into several workflow systems. Through the auto-descriptive capabilities of OpenMS' command line module used in all tools, it facilitates automated generation of wrappers for graphical user interface (GUI)-based (KNIME, Galaxy[1], gUSE[64]) workflow systems and integrates seamlessly into script-based workflow frameworks (e.g., nextflow[32]). This allows quick modification and prototyping of tools to update and extend existing workflows e.g., through daily updates of the wrapped tools based on the dynamic GitHub-hosted code base.

Role of OpenMS in this Thesis

As a Ph.D. student in the lab of the original authors of OpenMS, the author has been an OpenMS core developer and was responsible for maintenance, developer and user support, administration of development efforts as well as release management. The EPIFANY and ProteomicsLFQ TOPP tools introduced in this thesis, along with numerous bug fixes, data structures and a new external inference engine co-developed in an earlier version, have been contributed by the author and were incorporated into OpenMS. Figure 2.10 provides an overview on the structure of OpenMS and highlights relevant parts for this thesis.

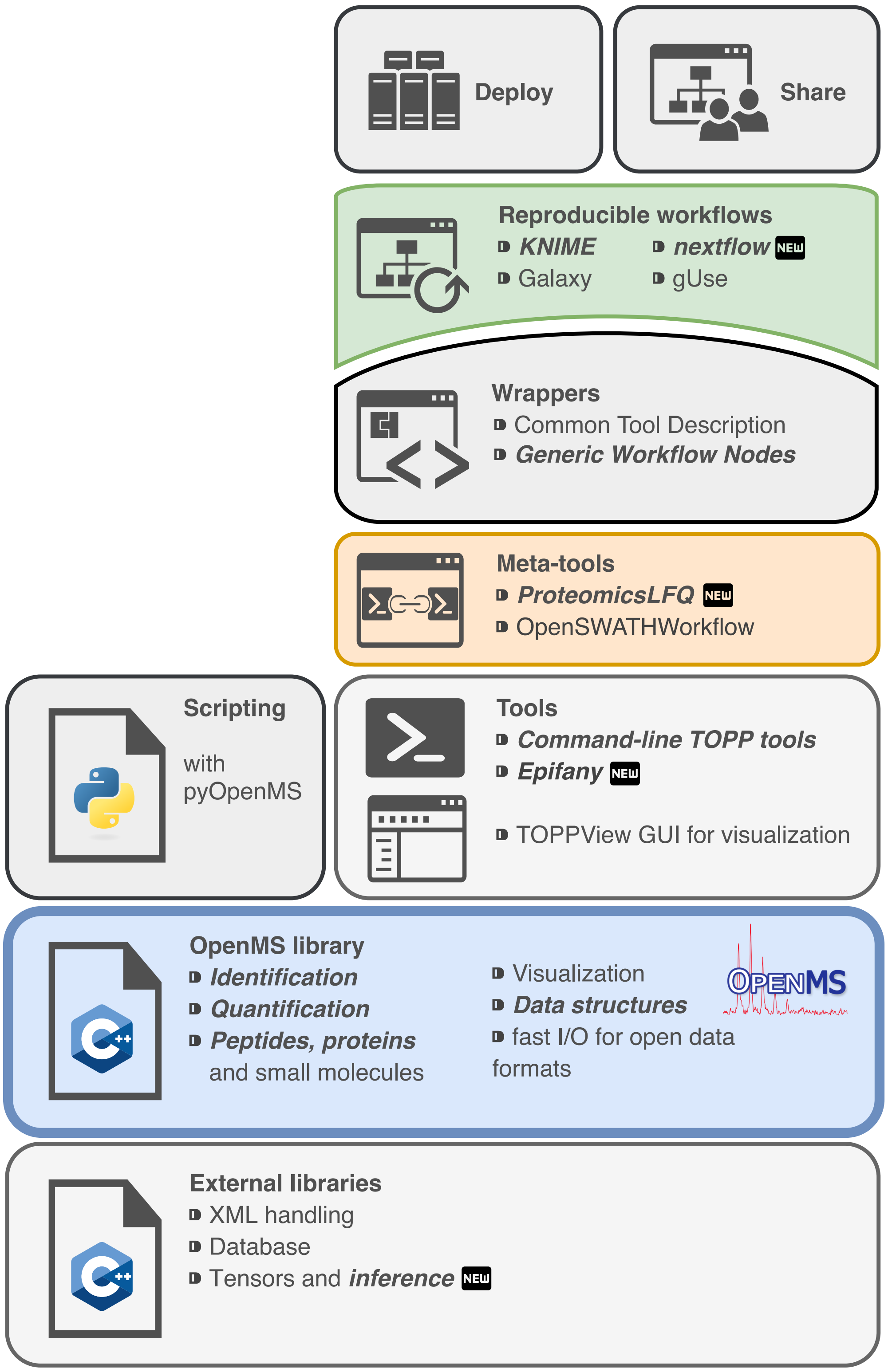


Figure 2.10: The structure of OpenMS. Changes made in the thesis are highlighted (bold for significantly changed, `NEW` for completely novel functionality/tools.)

Chapter 3

Efficient max-product inference for sums of random variables

Exact inference on discrete Bayesian networks in general is NP-hard (see **Section 2.4.5**). For networks used in protein inference, this intractability mostly comes from the potentially large number of parent proteins sharing a peptide. With certain model assumptions and parameterizations, each of those large converging subnetworks can be split into a random variable for the sum of present proteins and another linearly growing conditional probability table on the probability of a peptide given the sum of present proteins. Since sums of random variables can be calculated as convolutions, it can be efficiently solved through convolution trees for both sum-product[141] and approximate max-product inference[137]. However, current methods used for max-product inference on sums of random variables (equivalent to max-convolution) suffer from numerical instabilities[137]. Therefore, a revised method was designed and implemented as described in this chapter.

The remaining content of this chapter and corresponding Appendix A is part of the manuscript:

A Bounded p-norm Approximation of Max-Convolution for Sub-Quadratic Bayesian Inference on Additive Factors

Julianus Pfeuffer, Oliver Serang

Journal of Machine Learning Research 17, 1–39 (2016)

Author	Author position	Scientific ideas %	Data generation %	Analysis & interpretation %	Paper writing %
J. Pfeuffer	1	30	50	60	60
O. Serang	2	70	50	40	40
Status:	published				

3.1 Introduction

Max-convolution occurs frequently in signal processing and Bayesian inference: it is used in image analysis[126], in network calculus[12], in economic equilibrium analysis[154], and in a probabilistic variant of combinatoric generating functions, wherein information on a sum of values into their most probable constituent parts (e.g., identifying proteins from mass spectrometry[141,143]). Max-convolution operates on the semi-ring $(\max, \times)$, meaning that it behaves identically to a standard convolution, except it employs a max operation in lieu of the + operation in standard convolution (max-convolution is also equivalent to min-convolution, also called infimal convolution, which operates on the tropical semi-ring $(\min, +)$). Due to the importance and ubiquity of max-convolution, substantial effort has been invested into highly optimized implementations (e.g., implementations of the quadratic method on GPUs[173]).

Max-convolution can be defined using vectors (or discrete random variables, whose probability mass functions are analogous to nonnegative vectors) with the relationship $M = L + R$. Given the target sum $M = m$, the max-convolution finds the largest value $L[\ell] \times R[r]$ for which $m = \ell + r$.

$$\begin{aligned}
M[m] &= \max_{\ell,r:\, m=\ell+r} L[\ell]R[r] \\
&= \max_{\ell} L[\ell]R[m-\ell] \\
&= (L *_{\max} R)[m]
\end{aligned}$$

where $*_{\max}$ denotes the max-convolution operator. In probabilistic terms, this is equivalent to finding the highest probability of the joint events $\Pr(L = \ell, R = r)$ that would produce each possible value of the sum $M = L + R$ (note that in the probabilistic version, the vector M would subsequently need to be normalized so that its sum is 1).

Although applications of max-convolution are numerous, only a small number of methods exist for solving it[137]. These methods fall into two main categories, each with their own drawbacks: The first category consists of very accurate methods that are have worst-case runtimes either quadratic[14] or slightly more efficient than quadratic in the worst-case[13]. Conversely, the second type of method computes a numerical approximation to the desired result, but in $O(k \log_2(k))$ steps; however, no bound for the numerical accuracy of this method has been derived[137].

While the two approaches from the first category of methods for solving max-convolution do so by either using complicated sorting routines or by creating a bijection to an optimization problem, the numerical approach solves max-convolution by showing an equivalence between $*_{\max}$ and the process of first generating a vector $u^{(m)}$ for each index m of the result (where $u^{(m)}[\ell] = L[\ell]R[m-\ell]$ for all in-bounds indices) and subsequently computing the maximum $M[m] = \max_\ell u^{(m)}[\ell]$. When L and R are nonnegative, the maximization over the vector $u^{(m)}$ can be computed exactly via the Chebyshev norm

$$\begin{aligned} M[m] &= \max_\ell u^{(m)}[\ell] \\ &= \lim_{p\to\infty} \|u^{(m)}\|_p \end{aligned}$$

but requires $O(k^2)$ steps (where k is the length of vectors L and R). However, once a fixed p^*-norm is chosen, the approximation corresponding to that p^* can be computed by expanding the p^*-norm to yield

$$\begin{aligned} \lim_{p\to\infty} \|u^{(m)}\|_p &= \lim_{p\to\infty} \left(\sum_\ell \left(u^{(m)}[\ell]\right)^p \right)^{\frac{1}{p}} \\ &\approx \left(\sum_\ell \left(u^{(m)}[\ell]\right)^{p^*} \right)^{\frac{1}{p^*}} \\ &= \left(\sum_\ell L[\ell]^{p^*} \, R[m-\ell]^{p^*} \right)^{\frac{1}{p^*}} \\ &= \left(\sum_\ell \left(L^{p^*}\right)[\ell] \, \left(R^{p^*}\right)[m-\ell] \right)^{\frac{1}{p^*}} \\ &= \left(L^{p^*} * R^{p^*}\right)^{\frac{1}{p^*}}[m] \end{aligned}$$

where $L^{p^*} = \langle\, (L[0])^{p^*}, (L[1])^{p^*}, \ldots, (L[k-1])^{p^*} \,\rangle$ and $*$ denotes standard convolution. The standard convolution can be done via fast Fourier transform (FFT) in $O(k\log_2(k))$ steps, which is substantially more efficient than the $O(k^2)$ required by the naive method (**Algorithm 1**).

To date, the numerical method has demonstrated the best speed-accuracy trade-off on Bayesian inference tasks, and can be generalized to multiple dimensions (i.e., tensors). In particular, they have been used with probabilistic convolution trees[141] to efficiently compute the most probable values of discrete random variables $X_0, X_1, \ldots X_{n-1}$

for which the sum is known $X_0 + X_1 + \ldots X_{n-1} = y$. The one-dimensional variant of this problem (i.e., where each X_i is a one-dimensional vector) solves the probabilistic generalization of the subset sum problem, while the two-dimensional variant (i.e., where each X_i is a one-dimensional matrix) solves the generalization of the knapsack problem (note that these problems are not NP-hard in this specific case, because we assume an evenly-spaced discretization of the possible values of the random variables).

However, despite the practical performance that has been demonstrated by the numerical method, only cursory analysis has been performed to formalize the influence of the value of p^* on the accuracy of the result and to bound the error of the p^*-norm approximation. Optimizing the choice of p^* is non-trivial: Larger values of p^* more closely resemble a true maximization under the p^*-norm, but result in underflow (note that in **Algorithm 1**, the maximum values of both L and R can be divided out and then multiplied back in after max-convolution so that overflow is not an issue). Conversely, smaller values of p^* suffer less underflow, but compute a norm with less resemblance to maximization. Here we perform an in-depth analysis of the influence of p^* on the accuracy of numerical max-convolution, and from that analysis we construct a modified piecewise algorithm, on which we demonstrate bounds on the worst-case absolute error. This modified algorithm, which runs in $O(k \log(k) \log(\log(k)))$ steps, is demonstrated using a hidden Markov model describing the relationship between U.S. unemployment and the S&P 500 stock index.

We then extend the modified algorithm and introduce a second modified algorithm, which not only uses a single p-norm as a means of approximating the Chebyshev norm, but instead uses a sequence of p-norms and assembles them using a projection as a means to approximate the Chebyshev norm. Using numerical simulations as evidence, we make a conjecture regarding the relative error of the null space projection method. In practice, this null space projection algorithm is shown to have similar runtime and higher accuracy when compared with the piecewise algorithm.

3.2 Methods

We begin by outlining and comparing three numerical methods for max-convolution. By analyzing the benefits and deficits of each of these methods, we create improved variants. All of these methods will make use of the basic numerical max-convolution idea summarized in the introduction, and as such we first declare a method for computing the numerical max-convolution estimate for a given p^*. We call the algorithm `numericalMaxConvolveGivenPStar` (**Algorithm 1**).

Algorithm 1 Numerical max-convolution given a fixed p^*, a numerical method to estimate the max-convolution of two PMFs or nonnegative vectors. The parameters are two nonnegative vectors L' and R' (both scaled so that they have maximal element 1) and the numerical value p^* used for computation. The return value is a numerical estimate of the max-convolution $L' *_{\max} R'$.

```
1: procedure NUMERICALMAXCONVOLVEGIVENPSTAR($L', R', p^*$)
2:     $\forall \ell,\ vL[\ell] \leftarrow L[\ell]^{p^*}$
3:     $\forall r,\ vR[r] \leftarrow R[r]^{p^*}$
4:     $vM \leftarrow vL * vR$                    ▷ Standard FFT convolution is used here
5:     $\forall m,\ M'[m] \leftarrow vM[m]^{\frac{1}{p^*}}$
6:     return $M'$
7: end procedure
```

3.2.1 Fixed low-value p^* method

With a low p^* (e.g., 8) the effects of underflow will be minimal (as it is not very far from standard FFT convolution, an operation with high numerical stability), but it can still be imprecise due to numerical "bleed-in" (i.e., error due to contributions from non-maximal terms for a given $u^{(m)}$ because the p^*-norm is not identical to the Chebyshev norm). Overall, this will perform well on indices where the exact value of the result is small, but perform poorly when the exact value of the result is large.

3.2.2 Fixed high-value p^* method

In contrast to the method above, a fixed high p^* will offer the converse pros and cons: numerical artifacts due to bleed-in will be smaller (thus achieving greater performance on indices where the exact values of the result are larger), but underflow may be significant (and therefore, indices where the exact results of the max-convolution are small will be inaccurate).

3.2.3 Higher-order piecewise method

The higher-order piecewise method formalizes the empirical cutoff values found by Serang [137]; previously, numerical stability boundaries were found for each p^* by computing both the exact max-convolution (via the naive $O(k^2)$ method) and via the numerical method using the ascribed value of p^*, and finding the value below which the numerical values experienced a high increase in relative absolute error.

Those previously observed empirical numerical stability boundaries can be formalized by using the fact that the employed `numpy` implementation of FFT convolution has high accuracy on indices where the result has a value $\geq \tau$ relative to the maximum value; therefore, if the arguments L and R are both normalized so that each has a maximum value of 1, the fast max-convolution approximation is numerically stable for any index m where the result of the FFT convolution, i.e., $vM[m]$, is $\geq \tau$. The numpy documentation defines a conservative numeric tolerance for underflow $\tau = 10^{-12}$, which is a conservative estimate of the numerical stability boundary demonstrated in **Figure 3.1** (those boundary points occur very close to the true machine precision $\epsilon \approx 10^{-15}$).

Because Cooley-Tukey implementations of FFT-based convolution (e.g., the numpy implementation) are widely applied to large problems with extremely small error, we will make a simplification and assume that, when constraining the FFT result to reach a value higher than machine epsilon (+ tolerance threshold), the error from the FFT is negligible in comparison to the error introduced by the p^*-norm approximation. This is firstly because the only source of numerical error during FFT (assuming an FFT implementation with numerically precise twiddle factors) on vectors in $[0,1]^k$ will be the result of underflow from repeated addition and subtraction (neglecting the non-influencing multiplication with twiddle factors, which each have magnitude 1). The numerically imprecise routines are thus limited to $(x+y)-x$; when $x >> y$ (i.e., $\frac{y}{x} < \epsilon \approx 10^{-15}$, the machine precision), then $(x+y)-x$ will return 0 instead of y. To recover at least one bit of the significand, the intermediate results of the FFT must surpass machine precision ϵ (since the worst case addition initially happens with the maximum $x = 1.0$).

The maximum sum of any values from a list of k such elements can never exceed k; for this reason, a conservative estimate of the numerical tolerance (with regard to underflow) of a DFT (discrete Fourier transform) will be the smallest value of y for which $\frac{y}{k} > \epsilon$; thus, $y > \epsilon k$. This yields a conservative estimate of the minimum value in one index at the result of an DFT convolution: when the result at some index m is $> \epsilon k$, then the result should be numerically stable. We emphasize on the very conservative nature of this estimate for a minimum stable value that is derived under the assumption of a simple, naive implementation of a DFT. In general however, FFT (a special case of DFT algorithms) implementations (especially well-proven implementations of the Cooley-Tukey algorithm) tend to follow error bounds logarithmic in k[132], due to their divide-and-conquer strategies, which perform fewer arithmetic operations. In practice, even for longer vectors (e.g., $k = 1024$ as shown in **Figure 3.1**), it is empirically demonstrated that the influence of the length on the τ threshold is negligible. By using a numerical tolerance $\tau = 10^{-12}$, we ensure that the vast majority of numerical errors for the numerical max-convolution algorithm is due to the p^*-norm approximation (i.e., employing $\|u^{(m)}\|_{p^*}$ instead of $\|u^{(m)}\|_{\infty}$) and not due to the long-used and numerically performant FFT result. Furthermore, in practice the mean squared error due to FFT will

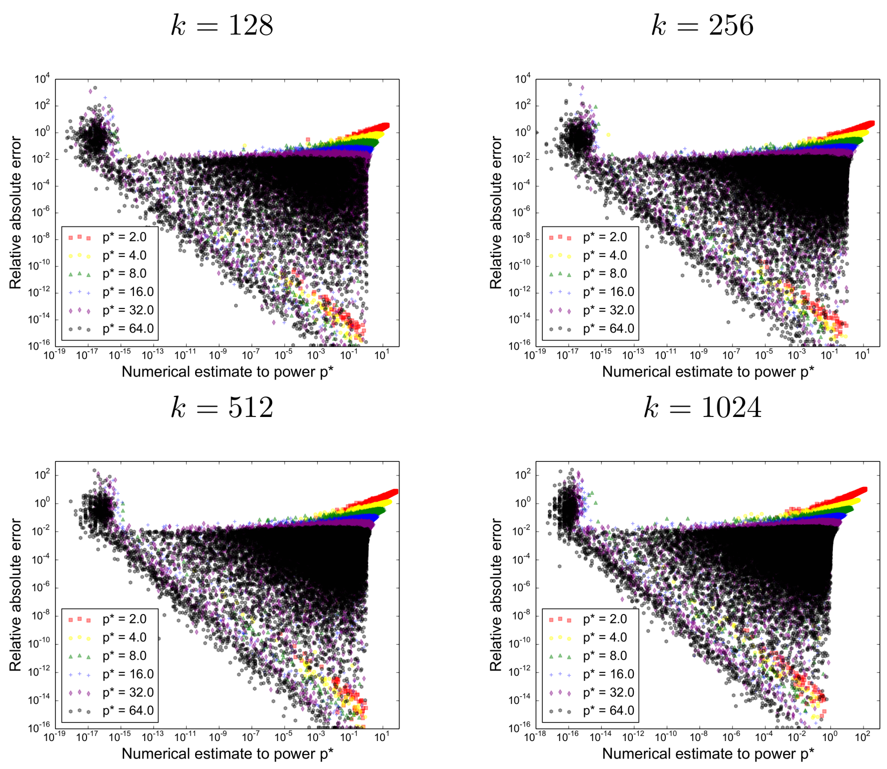


Figure 3.1: Empirical estimate of τ to construct a piecewise method. For each $k \in \{128, 256, 512, 1024\}$, 32 replicate max-convolutions (on vectors filled with uniform values) are performed. Error from two sources can be seen: error due to underflow is depicted in the sharp left mode, whereas error due to imperfect approximation, where $\|\cdot\|_{p^*} > \|\cdot\|_\infty$ can be seen in the gradual mode on the right. Error due to p^*-norm approximation is significantly smaller when p^* is larger (thereby flattening the right mode), but larger p^* values are more susceptible to underflow, pushing more indices into the left mode. Regardless of the value of k, error due to underflow occurs when $\left(\|\cdot\|_{p^*}\right)^{p^*}$ goes below $\approx 10^{-15}$; this is approximately the numerical tolerance for τ described by the numpy documentation. Therefore, at each index m we can construct a piecewise method that uses the largest value of p^* for which the FFT convolution result is not close to the machine precision (i.e., $(\|u^{(m)}\|_{p^*})^{p^*} \geq \tau$ for some $\tau > 10^{-15}$).

be much smaller than the conservative worst-case outlined here, because it is difficult for the largest intermediate summed value (in this case x) to be consistently large when many such very small values (in this case y) are encountered in the same list. Although τ could be chosen specifically for a problem of size k, note that this simple derivation is very conservative and thus it would be better to use a tighter bound for choosing τ. Regardless, for an FFT implementation that is not as performant (e.g., because it uses `float` types instead of `double`), increasing τ slightly would suffice.

Therefore, from this point forward we consider that the dominant cause of error to come from the max-convolution approximation. Using larger p^* values will provide a closer approximation; however, using a larger value of p^* may also drive values to zero (because the inputs L and R will be normalized within **Algorithm 1** so that the maximum of each is 1 when convolved via FFT), limiting the applicability of large p^* to indices m for which $vM[m] \geq \tau$.

Through this lens, the choice of p^* can be characterized by two opposing sources of error: higher p^* values better approximate $\|u^{(m)}\|_{p^*}$ but will be numerically unstable for many indices; lower p^* values provide worse approximations of $\|u^{(m)}\|_{p^*}$ but will be numerically unstable for only few indices. These opposing sources of error pose a natural method for improving the accuracy of this max-convolution approximation. By considering a small collection of p^* values, we can compute the full numerical estimate (at all indices) with each p^* using **Algorithm 1**; computing the full result at a given p^* is $\in O(k \log_2(k))$, so doing so on some small number c of p^* values considered, then the overall runtime will be $\in O(ck \log_2(k))$. Then, a final estimate is computed at each index by using the largest p^* that is stable (with respect to underflow) at that index. Choosing the largest p^* (of those that are stable with respect to underflow) corresponds to minimizing the bleed-in error, because the larger p^* becomes, the more muted the non-maximal terms in the norm become (and thus the closer the p^*-norm becomes to the true maximum).

Here we introduce this piecewise method and compare it to the simpler low-value $p^* = 8$ and high-value $p^* = 64$ methods and analyze the worst-case error of the piecewise method.

Algorithm 2 Piecewise numerical max-convolution , a numerical method to estimate the max-convolution of nonnegative vectors (revised to reduce bleed-in error). This procedure uses a p^* close to the largest possible stable value at each result index. The return value is a numerical estimate of the max-convolution $L *_{\max} R$. The runtime is in $O(k \log_2(k) \log_2(p^*_{\max}))$.

```
1: procedure NUMERICALMAXCONVOLVEPIECEWISE($L$, $R$, $p^*_{\max}$)
2:     $\ell_{\max} \leftarrow \text{argmax}_\ell\, L[\ell]$
3:     $r_{\max} \leftarrow \text{argmax}_r\, R[r]$
4:     $L' \leftarrow \frac{L}{L[\ell_{\max}]}$
5:     $R' \leftarrow \frac{R}{R[r_{\max}]}$                          ▷ Scale to a proportional problem on $L'$,$R'$
6:     allPStar $\leftarrow [2^0, 2^1, \ldots, 2^{\lceil \log_2(p^*_{\max}) \rceil}]$
7:     for $i \in \{0, 1, \ldots$ len(allPStar)$\}$ do
8:         resForAllPStar[$i$] $\leftarrow$ fftNonnegMaxConvolveGivenPStar($L'$, $R'$, allPStar[$i$])
9:     end for
10:    for $m \in \{0, 1, \ldots$ len($L$) + len($R$) $- 1\}$ do
11:        maxStablePStarIndex[$m$] $\leftarrow \max\{i :$ (resForAllPStar[$i$][$m$])$^{\text{allPStar}[i]} \geq \tau)\}$
12:    end for
13:    for $m \in \{0, 1, \ldots$ len($L$) + len($R$) $- 1\}$ do
14:        $i \leftarrow$ maxStablePStarIndex[$m$]
15:        result[$m$] $\leftarrow$ resForAllPStar[$i$][$m$]
16:    end for
17:    return $L[\ell_{\max}] \times R[r_{\max}] \times$ result                ▷ Undo previous scaling
18: end procedure
```

3.2.4 Availability

Code for exact max-convolution and the fast numerical methods (which includes piecewise, piecewise-corrected and null space projection methods) is implemented in Python and available at
`https://bitbucket.org/orserang/fast-numerical-max-convolution`.
All included code works for `numpy` arrays of any dimension, i.e., tensors.

3.3 Results

This section derives theoretical error bounds as well as a practical comparison on an example for the standard piecewise method. Furthermore, the development of an improvement with affine scaling is shown. Eventually, an evaluation of the latter is performed on a larger problem in which applied our technique to compute the Viterbi path for a hidden Markov model (HMM) to assess runtime and the level of error propagation.

3.3.1 Error and runtime analysis of the piecewise Method

We first analyze the error for a particular underflow-stable p^* and then use that to generalize to the piecewise method, which seeks to use the highest underflow-stable p^*.

Error analysis for a fixed underflow-stable p^*

We first scale L and R into L' and R' respectively, where the maximum elements of both L' and R' are 1; the absolute error can be found by unscaling the absolute error of the scaled problem:

$$|\text{exact}(L,R)[m]-\text{numeric}(L',R')[m]| \\ = \max_{\ell} L[\ell]\ \max_{r} R[r]\ |\text{exact}(L',R')[m]-\text{numeric}(L',R')[m]|.$$

We first derive an error bound for the scaled problem on L',R' (any mention of a vector $u^{(m)}$ refers to the scaled problem), and then reverse the scaling to demonstrate the error bound on the original problem on L,R.

For any particular "underflow-stable" p^* (i.e., any value of p^* for which $\left(\|u^{(m)}\|_{p^*}\right)^{p^*} \geq \tau$), the absolute error for the numerical method for fast max-convolution can be bound fairly easily by factoring out the maximum element of $u^{(m)}$ (this maximum element is equivalent to the Chebyshev norm) from the p^*-norm:

$$\begin{aligned}
&|\text{exact}(L',R')[m] - \text{numeric}(L',R')[m]| \\
&= \left| \|u^{(m)}\|_{p^*} - \|u^{(m)}\|_\infty \right| \\
&= \|u^{(m)}\|_{p^*} - \|u^{(m)}\|_\infty \\
&= \|u^{(m)}\|_\infty \left(\frac{\|u^{(m)}\|_{p^*}}{\|u^{(m)}\|_\infty} - 1 \right) \\
&= \|u^{(m)}\|_\infty \left(\left\| \frac{u^{(m)}}{\|u^{(m)}\|_\infty} \right\|_{p^*} - 1 \right) \\
&= \|u^{(m)}\|_\infty \left(\|v^{(m)}\|_{p^*} - 1 \right)
\end{aligned}$$

where $v^{(m)}$ is a nonnegative vector of the same length as $u^{(m)}$ (this length is denoted k_m) where $v^{(m)}$ contains one element equal to 1 (because the maximum element of $u^{(m)}$ must, by definition, be contained within $u^{(m)}$) and where no element of $v^{(m)}$ is greater than 1 (also provided by the definition of the maximum). These observations result in the equation

$$\begin{aligned}
\|v^{(m)}\|_{p^*} &\leq \|(1,1,\ldots 1)\|_{p^*} \\
&= \left(\sum_i^{k_m} 1^{p^*} \right)^{\frac{1}{p^*}} \\
&= {k_m}^{\frac{1}{p^*}}.
\end{aligned}$$

Thus, since $\|v^{(m)}\|_{p^*} \geq 1$, the error is bounded:

$$\begin{aligned}
&|\text{exact}(L',R')[m] - \text{numeric}(L',R')[m]| \\
&= \|u^{(m)}\|_\infty \left(\|v^{(m)}\|_{p^*} - 1 \right) \\
&\leq \|v^{(m)}\|_{p^*} - 1 \\
&\leq k_m^{\frac{1}{p^*}} - 1,
\end{aligned}$$

because $\forall m,\ \|u^{(m)}\|_\infty \leq 1$ for a scaled problem on L',R'.

Error analysis of piecewise method

However, the bounds derived above are only applicable for p^* where $\|u^{(m)}\|_{p^*}^{p^*} \geq \tau$. The piecewise method is slightly more complicated, and can be partitioned into two cases: In the first case, the top contour is used (i.e., when p^*_{max} is underflow-stable). Conversely, in the second case, a middle contour is used (i.e., when p^*_{max} is not underflow-stable). In this context, in general a contour comprises of a set of indices m with the same maximum stable p^*.

In the first case, when we use the top contour $p^* = p^*_{\text{max}}$, we know that p^*_{max} must be underflow-stable, and thus we can reuse the bound given an underflow-stable p^*.

In the second case, because the p^* used is $< p^*_{\text{max}}$, it follows that the next higher contour (using $2p^*$) must not be underflow-stable (because the highest underflow-stable p^* is used and because the p^* are searched in log-space). The bound derived above that demonstrated

$$\|u^{(m)}\|_{p^*} \leq \|u^{(m)}\|_\infty k_m^{\frac{1}{p^*}}$$

can be combined with the property that $\|\cdot\|_{p^*} \geq \|\cdot\|_\infty$ for any $p^* \geq 1$ to show that

$$\|u^{(m)}\|_\infty \in \left[\frac{\|u^{(m)}\|_{p^*}}{k_m^{\frac{1}{p^*}}}, \|u^{(m)}\|_{p^*} \right].$$

Thus the absolute error can also be shown to be bounded using the fact that we are in a middle contour:

$$\begin{aligned}
&= \|u^{(m)}\|_{p^*} - \|u^{(m)}\|_\infty \\
&= \|u^{(m)}\|_{p^*} \left(1 - \frac{\|u^{(m)}\|_\infty}{\|u^{(m)}\|_{p^*}} \right) \\
&\leq \|u^{(m)}\|_{p^*} \left(1 - k_m^{\frac{-1}{p^*}} \right) \\
&< \tau^{\frac{1}{2p^*}} \left(1 - k_m^{\frac{-1}{p^*}} \right).
\end{aligned}$$

The absolute error from middle contours will be quite small when $p^* = 1$ is the maximum underflow-stable value of p^* at index m, because $\tau^{\frac{1}{2p^*}}$, the first factor in the error bound, will become $\sqrt{\tau} \approx 10^{-6}$, and $1 - k_m^{\frac{-1}{p^*}} < 1$ (qualitatively, this indicates that a small p^* is only used when the result is very close to zero, leaving little room for absolute error). Likewise, when a very large p^* is used, then $1 - k_m^{\frac{-1}{p^*}}$ becomes very small, while $\tau^{\frac{1}{2p^*}} < 1$ (qualitatively, this indicates that when a large p^* is used, the $\|\cdot\|_{p^*} \approx \|\cdot\|_\infty$, and thus there is little absolute error). Thus, for the extreme values of p^*, middle contours will produce fairly small absolute errors. The unique mode p^*_{mode} can be found by finding the value that solves

$$\frac{\partial}{\partial p^*_{\text{mode}}} \left(\tau^{\frac{1}{2p^*_{\text{mode}}}} \left(1 - k_m^{\frac{-1}{p^*_{\text{mode}}}} \right) \right) = 0,$$

which yields

$$p^*_{\text{mode}} = \frac{\log_2(k_m)}{\log_2(-\frac{2\log_2(k) - \log_2(\tau)}{\log_2(\tau)})}.$$

An appropriate choice of $p^*_{\max}$ should be $> p^*_{\text{mode}}$ so that the error for any contour (both middle contours and the top contour) is smaller than the error achieved at p^*_{mode}, allowing us to use a single bound for both. Choosing $p^*_{\max} = p^*_{\text{mode}}$ would guarantee that all contours are no worse than the middle-contour error at p^*_{mode}; however, using $p^*_{\max} = p^*_{\text{mode}}$ is still quite liberal, because it would mean that for indices in the highest contour (there must be a nonempty set of such indices, because the scaling on L' and R' guarantees that the maximum index will have an exact value of 1, meaning that the approximation endures no underflow and is underflow-stable for every p^*), a better error *could* be achieved by increasing $p^*_{\max}$. For this reason, we choose $p^*_{\max}$ so that the top-contour error produced at $p^*_{\max}$ is not substantially larger than all errors produced for p^* before the mode (i.e., for $p^* < p^*_{\text{mode}}$).

Choosing any value of $p^*_{\max} > p^*_{\text{mode}}$ guarantees the worst-case absolute error bound derived here; however, increasing $p^*_{\max}$ further over p^*_{mode} may possibly improve the mean squared error in practice (because it is possible that many indices in the result would be numerically stable with p^* values substantially larger than p^*_{mode}). However, increasing $p^*_{\max} >> p^*_{\text{mode}}$ will produce diminishing returns and generally benefit only a very small number of indices in the result, which have exact values very close to 1. In order to balance these two aims (increasing $p^*_{\max}$ enough over p^*_{mode} but not excessively so), we make a qualitative assumption that a non-trivial number of indices require us to use a p^* below p^*_{mode}; therefore, increasing $p^*_{\max}$ to produce an error significantly smaller than the lowest worst-case error for contours below the mode (i.e., $p^* < p^*_{\text{mode}}$) will increase the runtime without significantly decreasing the mean squared error (which will become dominated by the errors from indices that use $p^* < p^*_{\text{mode}}$). The lowest worst-case error contour below the mode is $p^* = 1$ (because the absolute error function is unimodal, and thus must be increasing until p^*_{mode} and decreasing afterward); therefore, we heuristically specify that $p^*_{\max}$ should produce a worst-case error on a similar order of magnitude to the worst-case error produced with $p^* = 1$. In practice, specifying the errors at $p^*_{\max}$ and $p^* = 1$ should be equal is very conservative (it produces very large estimates of $p^*_{\max}$, which sometimes benefit only one or two indices in the result); for this reason, we heuristically choose that the worst-case error at $p^*_{\max}$ should be no worse than square root of the worst case error at $p^* = 1$ (this makes the choice of $p^*_{\max}$ less conservative because the errors at $p^* = 1$ are very close to zero, and thus their square root is larger). The square root was chosen because it produced, for the applications described in this paper, the smallest value of $p^*_{\max}$ for which the mean squared error was significantly lower than using $p^*_{\max} = p^*_{\text{mode}}$ (the lowest value of $p^*_{\max}$ guaranteed to produce the absolute error bound). This heuristic does satisfy the worst-case bound outlined here (because, again, $p^*_{\max} > p^*_{\text{mode}}$), but it could be substantially improved if an expected distribution of magnitudes in the result vector were known ahead of time: prior knowledge regarding the number of points stable at each p^* considered would enable a well-motivated choice of $p^*_{\max}$ that truly optimizes the expected mean squared error.

From this heuristic choice of $p^*_{\max}$, solving

$$\sqrt{\sqrt{\tau}\left(1-\frac{1}{k}\right)} = k^{\frac{1}{p^*_{\max}}} - 1$$

(with the square root of the worst-case at $p^* = 1$ on the left and the worst-case error at $p^*_{\max}$ on the right) yields

$$\begin{aligned} p^*_{\max} &= \frac{\log_2(k)}{\log_2(1+\sqrt{\sqrt{\tau}\left(1-\frac{1}{k}\right)})} \\ &\approx \frac{\log_2(k)}{\log_2(1+\sqrt{\sqrt{\tau}})} \end{aligned}$$

for any non-trivial problem (i.e., when $k >> 1$), and thus

$$p^*_{\max} \approx \log_{1+\tau^{\frac{1}{4}}}(k),$$

indicating that the absolute error at the top contour will be roughly equal to the fourth root of τ.

Worst-case absolute error

By setting $p^*_{\max}$ in this manner, we guarantee that the absolute error at any index of any unscaled problem on L, R is less than

$$\max_\ell L[\ell] \ \max_r R[r] \ \tau^{\frac{1}{2p^*_{\text{mode}}}} \left(1 - k_m^{\frac{-1}{p^*_{\text{mode}}}}\right)$$

where p^*_{mode} is defined above. The full formula for the middle-contour error at this value of p^*_{mode} does not simplify and is therefore quite large; for this reason, it is not reported here, but this gives a numeric bound of the worst case middle-contour error that is bound in terms of the variable k (and with no other free variables).

Runtime analysis

The piecewise method clearly performs $\log_2(p^*_{\max})$ FFTs (each requiring $O(k \log_2(k))$ steps); therefore, since $p^*_{\max}$ is chosen to be $\log_{1+\tau^{\frac{1}{4}}}(k)$ (to achieve the desired error bound), the total runtime is thus

$$O(k \log_2(k) \log_2(\log_{1+\tau^{\frac{1}{4}}}(k)).$$

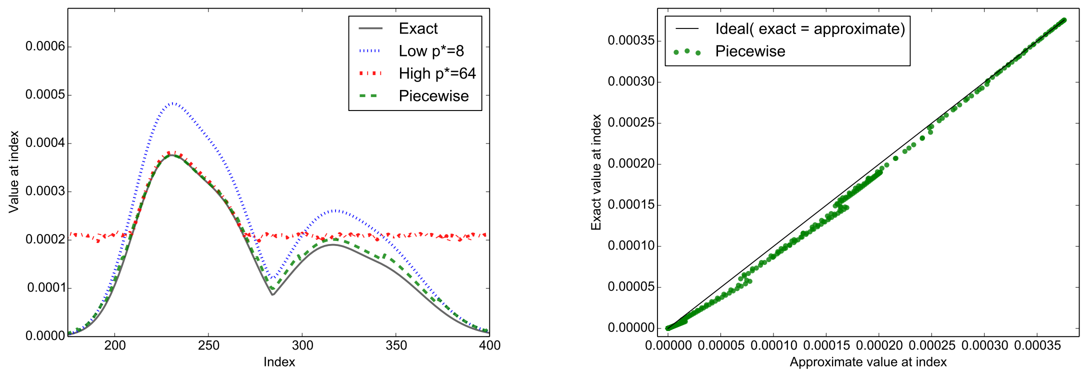
For any practically sized problem, the $\log_2(\log_{1+\tau^{\frac{1}{4}}}(k))$ factor is essentially a constant; even when k is chosen to be the number of particles in the observable universe ($\approx 2^{270}$;[36]), the $\log_2(\log_{1+\tau^{\frac{1}{4}}}(k))$ is ≈ 18, meaning that for any problem of practical size, the full piecewise method is no more expensive than computing between 1 and 18 FFTs.

3.3.2 Comparison of low-value $p^* = 8$, high-value $p^* = 64$, and piecewise method

We first use an example max-convolution problem to compare the results from the low-value $p^* = 8$, the high-value $p^* = 64$ and piecewise methods. At every index, these various approximation results are compared to the exact values, as computed by the naive quadratic method (**Figure 3.2a**).

3.3.3 Improved affine piecewise method

Figure 3.2b depicts a scatter plot of the exact result vs. the piecewise approximation at every index (using the same problem from **Figure 3.2a**). It shows a clear banding pattern: the exact and approximate results are clearly correlated, but each contour (i.e., each collection of indices that use a specific p^*) has a different average slope between



(a) (b)

Figure 3.2: The accuracy of numerical fast max-convolution methods. (a) Different approximations for a sample max-convolution problem. The low-p^* method is underflow-stable, but overestimates the result. The high-p^* method is accurate when underflow-stable, but experiences underflow at many indices. The piecewise method stitches together approximations from different p^* to maintain underflow-stability. **(b)** Exact vs. piecewise approximation at various indices of the same problem. A clear banding pattern is observed with one tight, elliptical cluster for each contour. The slope of the clusters deviates more for the contours using lower p^* values.

the exact and approximate values, with higher p^* contours showing a generally larger slope and smaller p^* contours showing greater spread and lower slopes. This intuitively makes sense, because the bounds on $\|u^{(m)}\|_\infty \in [\|u^{(m)}\|_{p^*} k_m^{\frac{-1}{p^*}}, \|u^{(m)}\|_{p^*}]$ derived above constrain the scatter plot points inside a quadrilateral envelope (**Figure 3.3**).

The correlations within each contour can be exploited to correct biases that emerge for smaller p^* values. In order to do this, $\|u^{(m)}\|_\infty$ must be computed for at least two points m_1 and m_2 within the contour, so that a mapping $\|u^{(m)}\|_{p^*} \approx f(\|u^{(m)}\|_{p^*}) = a\|u^{(m)}\|_{p^*} + b$ can be constructed. Fortunately, a single $\|u^{(m)}\|_\infty$ can be computed exactly in $O(k)$ (by actually computing a single $u^{(m)}$ and computing its max, which is equivalent to computing a single index result via the naive quadratic method). As long as the exact value $\|u^{(m)}\|_\infty$ is computed for only a small number of indices, the order of the runtime will not change (each contour already costs $O(k\log_2(k))$, so adding a small number of $O(k)$ steps for each contour will not change the asymptotic runtime).

If the two indices chosen are

$$m_{\min} = \operatorname*{argmin}_{m \in \text{contour}(p^*)} \|u^{(m)}\|_{p^*}$$

and

$$m_{\max} = \operatorname*{argmax}_{m \in \text{contour}(p^*)} \|u^{(m)}\|_{p^*},$$

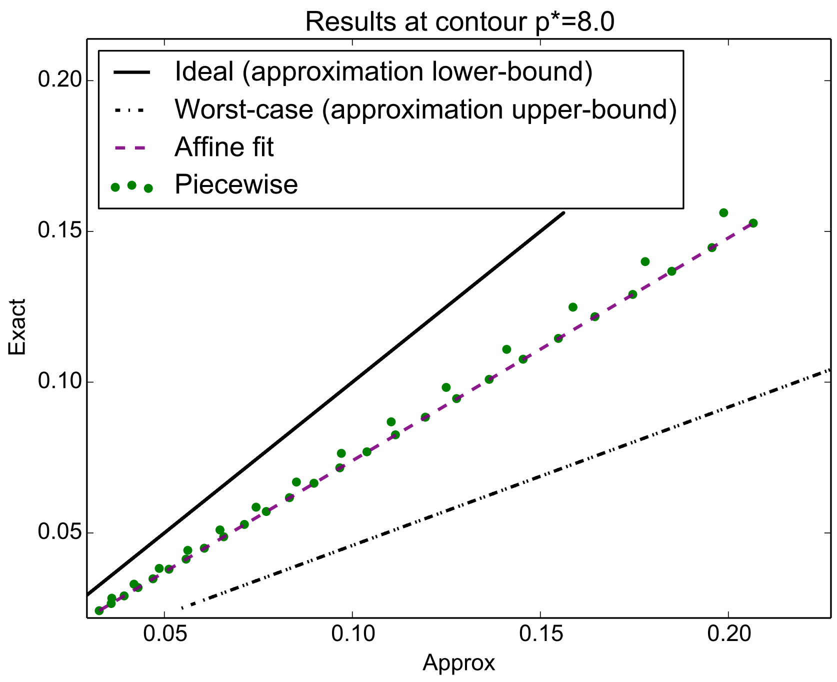


Figure 3.3: A single contour from the piecewise approximation. The cluster of points (one point for each index in the previous figure) is bounded by the exact value (ideal approximation) and the approximation upper-bound for $p^* = 8$ (worst-case approximation). The points are well described by an affine function fit using the left-most and right-most points.

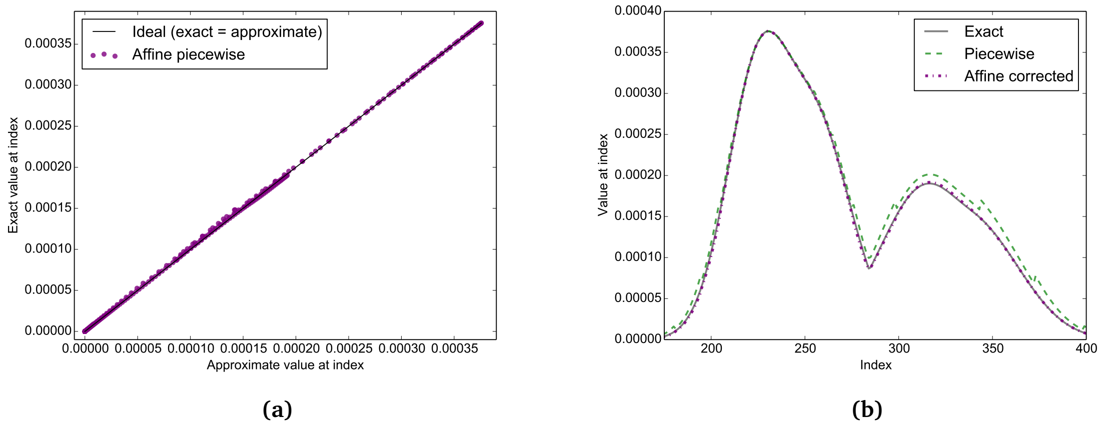


Figure 3.4: Piecewise method with affine contour fitting. The approximate values at each index of the max-convolution problem are almost identical to the exact result at the same index.

then we are guaranteed that the affine function f can be written as a convex combination of the exact values at those extreme points (using barycentric coordinates):

$$f(\|u^{(m)}\|_{p^*}) = \lambda_m \|u^{(m_{\max})}\|_\infty + (1-\lambda_m)\,\|u^{(m_{\min})}\|_\infty$$

$$\lambda_m = \frac{\|u^{(m)}\|_{p^*} - \|u^{(m_{\min})}\|_{p^*}}{\|u^{(m_{\max})}\|_{p^*} - \|u^{(m_{\min})}\|_{p^*}} \in [0,1]$$

Thus, by computing $\|u^{(m_{\min})}\|_\infty$ and $\|u^{(m_{\max})}\|_\infty$ (each in $O(k)$ steps), we can compute an affine function f to correct contour-specific trends (**Algorithm 3**).

Error analysis of improved affine piecewise method

By exploiting the convex combination used to define f, the absolute error of the affine piecewise method can also be bound. Qualitatively, this is because, by fitting on the extrema in the contour, we are now interpolating. If the two points used to determine the parameters of the affine function were not chosen in this manner to fit the affine function, then it would be possible to choose two points with very close x-values (i.e., similar approximate values) and disparate y-values (i.e., different exact values), and extrapolating to other points could propagate a large slope over a large distance; using the extreme points forces the affine function to be a convex combination of the extrema, thereby avoiding this problem.

Algorithm 3 Improved affine piecewise numerical max-convolution, a numerical method to estimate the max-convolution nonnegative vectors (further revised to reduce numerical error). This procedure uses a p^* close to the largest possible stable value at each result index. The return value is a numerical estimate of the max-convolution $L *_{\max} R$. The runtime is in $O(k \log_2(k) \log_2(p^*_{\max}))$.

```
1: procedure NUMERICALMAXCONVOLVEPIECEWISEAFFINE($L$, $R$, $p^*_{\max}$)
2:     $\ell_{\max} \leftarrow \mathrm{argmax}_\ell \, L[\ell]$
3:     $r_{\max} \leftarrow \mathrm{argmax}_r \, R[r]$
4:     $L' \leftarrow \frac{L}{L[\ell_{\max}]}$
5:     $R' \leftarrow \frac{R}{R[r_{\max}]}$                    ▷ Scale to a proportional problem on $L'$,$R'$
6:     allPStar $\leftarrow [2^0, 2^1, \ldots, 2^{\lceil \log_2(p^*_{\max}) \rceil}]$
7:     for $i \in \{0, 1, \ldots,$len(allPStar)$\}$ do
8:         resForAllPStar[$i$] $\leftarrow$ fftNonnegMaxConvolveGivenPStar($L'$, $R'$, allPStar[$i$])
9:     end for
10:    for $m \in \{0, 1, \ldots,$len($L$)+len($R$)$- 1\}$ do
11:        maxStablePStarIndex[$m$] $\leftarrow \max\{i :$ (resForAllPStar[$i$][$m$])$^{\text{allPStar}[i]} \geq \tau)\}$
12:    end for
13:    result $\leftarrow$ affineCorrect(resForAllPStar, maxStablePStarIndex)
14:    return $L[\ell_{\max}] \times R[r_{\max}] \times$ result                ▷ Undo previous scaling
15: end procedure
```

Algorithm 4 Subroutine for correcting errors in a contour, with an affine transformation based on exact boundary points. It needs the results of the evaluation of the different p-norms as well as the (index of the) maximum stable values of p^* at every index.

```
1: procedure AFFINECORRECT(resForAllPStar, maxStablePStarIndex)
2:     $\forall i$, slope[$i$] $\leftarrow 1$
3:     $\forall i$, bias[$i$] $\leftarrow 0$
4:     usedPStar $\leftarrow$ set(maxStablePStarIndex)
5:     for $i \in$ usedPStar do
6:         contour $\leftarrow \{m :$ maxStablePStarIndex[$m$] $= i\}$
7:         mMin $\leftarrow \mathrm{argmin}_{m \in \text{contour}}$ resForAllPStar[$i$][$m$]
8:         mMax $\leftarrow \mathrm{argmax}_{m \in \text{contour}}$ resForAllPStar[$i$][$m$]
9:         xMin $\leftarrow$ resForAllPStar[$i$][mMin]
10:        xMax $\leftarrow$ resForAllPStar[$i$][mMax]
11:        yMin $\leftarrow$ maxConvolutionAtIndex(mMin)
12:        yMax $\leftarrow$ maxConvolutionAtIndex(mMax)
13:        if xMax > xMin then
14:            slope[$i$] $\leftarrow \frac{\text{yMax}-\text{yMin}}{\text{xMax}-\text{xMin}}$
15:            bias[$i$] $\leftarrow$ yMin - slope[$i$] $\times$ xMin
16:        else
17:            slope[$i$] $\leftarrow \frac{\text{yMax}}{\text{xMax}}$
18:        end if
19:    end for
20:    for $m \in \{0, 1, \ldots,$ len($L$) + len($R$) $- 1\}$ do
21:        $i \leftarrow$ maxStablePStarIndex[$m$]
22:        result[$m$] $\leftarrow$ resForAllPStar[$i$][$m$] $\times$ slope[$i$] + bias[$i$]
23:    end for
24:    return result
25: end procedure
```

$$\begin{aligned}
& f(\|u^{(m)}\|_{p^*}) \\
& = \lambda_m \|u^{(m_{\max})}\|_\infty + (1-\lambda_m)\|u^{(m_{\min})}\|_\infty \\
& \in \left[\lambda_m \frac{\|u^{(m_{\max})}\|_{p^*}}{k_{m_{\max}}^{\frac{1}{p^*}}} + (1-\lambda_m) \frac{\|u^{(m_{\min})}\|_{p^*}}{k_{m_{\min}}^{\frac{1}{p^*}}}, \right. \\
& \qquad \left. \lambda_m \|u^{(m_{\max})}\|_{p^*} + (1-\lambda_m)\|u^{(m_{\min})}\|_{p^*} \right] \\
& \subseteq \left[\lambda_m \frac{\|u^{(m_{\max})}\|_{p^*}}{k^{\frac{1}{p^*}}} + (1-\lambda_m) \frac{\|u^{(m_{\min})}\|_{p^*}}{k^{\frac{1}{p^*}}}, \right. \\
& \qquad \left. \lambda_m \|u^{(m_{\max})}\|_{p^*} + (1-\lambda_m)\|u^{(m_{\min})}\|_{p^*} \right] \\
& = \left[k^{\frac{-1}{p^*}} \left(\lambda_m \|u^{(m_{\max})}\|_{p^*} + (1-\lambda_m)\|u^{(m_{\min})}\|_{p^*} \right), \right. \\
& \qquad \left. \lambda_m \|u^{(m_{\max})}\|_{p^*} + (1-\lambda_m)\|u^{(m_{\min})}\|_{p^*} \right] \\
& = \left[k^{\frac{-1}{p^*}} \|u^{(m)}\|_{p^*}, \|u^{(m)}\|_{p^*} \right]
\end{aligned}$$

The worst-case absolute error of the scaled problem on L', R' can be defined as

$$\max_m | f(\|u^{(m)}\|_{p^*}) - \|u^{(m)}\|_\infty |.$$

Because the function $f(\|u^{(m)}\|_{p^*}) - \|u^{(m)}\|_\infty$ is affine, it's derivative can never be zero, and thus Lagrangian theory states that the maximum must occur at a boundary point. Therefore, the worst-case absolute error is

$$\begin{aligned}
\leq & \quad \max\{\|u^{(m)}\|_{p^*} - \|u^{(m)}\|_\infty, \|u^{(m)}\|_\infty - \|u^{(m)}\|_{p^*} k^{\frac{-1}{p^*}}\} \\
= & \quad \|u^{(m)}\|_{p^*} - \|u^{(m)}\|_\infty,
\end{aligned}$$

which is identical to the worst-case error bound before applying the affine transformation f. Thus applying the affine transformation can dramatically improve error, but will not make it worse than the original worst-case.

3.3.4 An improved approximation of the Chebyshev norm

In order to improve the error of the piecewise numerical method, we consider the worst-case, when $\frac{u^{(m)}}{\|u^{(m)}\|_\infty} = (1, 1, \ldots 1)$. In this case, computing any two norms of $u^{(m)}$

(at p_1^* and p_2^*) would be sufficient to solve exactly for $\|u^{(m)}\|_\infty$, because

$$\begin{aligned} \|u^{(m)}\|_{p_1^*}^{p_1^*} &\propto \|u^{(m)}\|_\infty^{p_1^*} \\ \|u^{(m)}\|_{p_2^*}^{p_2^*} &\propto \|u^{(m)}\|_\infty^{p_2^*}, \end{aligned}$$

where the proportionality constant is $k_m = \mathtt{len}(u^{(m)})$.

Thus we see that although the p^*-norm approximation of the Chebyshev norm is a good approximation, the curve of the norms (at various different p^*) values holds far more information than a simple point estimate would. Therefore, we derive an improved algorithm by proceeding as follows: First, we note that, rather than computing the p^*-norm of a vector $u^{(m)}$ by summing all elements of $u^{(m)}$ taken to the power p^*, it is possible to equivalently sum over only the unique values of $u^{(m)}$ (here denoted in $\beta_1, \beta_2, \ldots$) taken to the p^* where each term in the sum is weighted by the number of occurrences of each (denoted $h_1, h_2, \ldots$, respectively). For this reason, we can then use a sequence of norms to compute a (potentially smaller) collection of approximate unique values $\alpha_1, \alpha_2, \ldots \alpha_r$, where once again, the p^*-norm is equal to the sum over those unique values to the p^*, where each term in the sum is weighted by numbers of occurrences $n_1, n_2, \ldots n_r$. Therefore, given $2r$ different norms of $u^{(m)}$, it is possible to project and estimate r unique α_i values. The maximum of those $\alpha_1, \alpha_2, \ldots \alpha_r$ values, (the $\alpha_1, \alpha_2, \ldots \alpha_r$ values can be thought of as projections of the true unique values $\beta_1, \beta_2, \ldots$) can then be used as an estimate of the true maximum element in $u^{(m)}$, which we will now demonstrate.

Specifically, where k_m is the length of $u^{(m)}$ and where there are $e_m \leq k_m$ unique values (β_i) in $u^{(m)}$, we can model the norms perfectly with

$$\|u^{(m)}\|_{p^*}^{p^*} = \sum_i^{e_m} h_i \beta_i^{p^*}$$

where h_i is an integer that indicates the number of times β_i occurs in $u^{(m)}$ (and where $\sum_i h_i = k_m = \mathtt{len}(u^{(m)})$). This multi-set view of the vector $u^{(m)}$ can be used to project it down to a dimension r:

$$\begin{bmatrix} \alpha_1^{p^*} & \alpha_2^{p^*} & & \alpha_r^{p^*} \\ \alpha_1^{2p^*} & \alpha_2^{2p^*} & \cdots & \alpha_r^{2p^*} \\ \alpha_1^{3p^*} & \alpha_2^{3p^*} & & \alpha_r^{3p^*} \\ \vdots & \vdots & & \vdots \\ \alpha_1^{\ell p^*} & \alpha_2^{\ell p^*} & & \alpha_r^{\ell p^*} \end{bmatrix} \cdot \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_r \end{bmatrix} = \begin{bmatrix} \|u^{(m)}\|_{p^*}^{p^*} \\ \|u^{(m)}\|_{2p^*}^{2p^*} \\ \|u^{(m)}\|_{3p^*}^{3p^*} \\ \vdots \\ \|u^{(m)}\|_{\ell p^*}^{\ell p^*} \end{bmatrix}.$$

By solving the above system of equations for all α_i, the maximum $\hat{\alpha} = \max_i \alpha_i$ can be used to approximate the true maximum $\max_i \beta_i = \|u^{(m)}\|_\infty$. This projection can be thought of as querying distinct moments of the distribution $\mathrm{pmf}_{U^{(m)}}$ that corresponds to some unknown vector $u^{(m)}$, and then assembling the moments into a model in order to predict the unknown maximum value in $u^{(m)}$. Of course, when r, the number of terms in our model, is sufficiently large, then computing r norms of $u^{(m)}$ will result in an exact result, but it could result in $O(k_m)$ execution time, meaning that our numerical max-convolution algorithm becomes quadratic; therefore, we must consider that a small number of distinct moments are queried in order to estimate the maximum value in $u^{(m)}$. Regardless, the system of equations above is quite difficult to solve directly via elimination for even very small values of r, because the symbolic expressions become quite large and because symbolic polynomial roots cannot be reliably computed when the degree of the polynomial is > 5. Even in cases when it can be solved directly, it will be far too inefficient.

For this reason, we solve for the α_i values using an exact, alternative approach: If we define a polynomial

$$\gamma(x) = \left(x - \alpha_1^{p^*}\right)\left(x - \alpha_2^{p^*}\right)\cdots\left(x - \alpha_r^{p^*}\right),$$

then

$$x \in \{\alpha_1^{p^*}, \alpha_2^{p^*}, \ldots, \alpha_r^{p^*}\} \Leftrightarrow \gamma(x) = 0.$$

We can expand $\gamma(x) = \gamma_0 + \gamma_1 x + \gamma_2 x^2 + \cdots + \gamma_r x^r$, and then write

$$\begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_r \end{bmatrix} \cdot \begin{bmatrix} \alpha_1^{p^*} & \alpha_2^{p^*} & & \alpha_r^{p^*} \\ \alpha_1^{2p^*} & \alpha_2^{2p^*} & \cdots & \alpha_r^{2p^*} \\ \alpha_1^{3p^*} & \alpha_2^{3p^*} & & \alpha_r^{3p^*} \\ \vdots & \vdots & & \vdots \\ \alpha_1^{\ell p^*} & \alpha_2^{\ell p^*} & & \alpha_r^{\ell p^*} \end{bmatrix} \cdot \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_r \end{bmatrix} =$$

$$\begin{bmatrix} \alpha_1^{p^*}\gamma(\alpha_1^{p^*}) & \alpha_2^{p^*}\gamma(\alpha_2^{p^*}) & \alpha_3^{p^*}\gamma(\alpha_3^{p^*}) & \cdots & \alpha_r^{p^*}\gamma(\alpha_r^{p^*}) \end{bmatrix} \cdot \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_r \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_r \end{bmatrix} = 0,$$

which indicates that

$$\begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_r \end{bmatrix} \cdot \begin{bmatrix} ||u^{(m)}||_{p^*}^{p^*} \\ ||u^{(m)}||_{2p^*}^{2p^*} \\ ||u^{(m)}||_{3p^*}^{3p^*} \\ \vdots \\ ||u^{(m)}||_{\ell p^*}^{\ell p^*} \end{bmatrix} = 0.$$

Furthermore, $\gamma(x) = 0, x \neq 0 \Leftrightarrow x^i\gamma(x) = 0, i \in \mathbb{N}$; therefore we can write

$$\begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_r & 0 & 0 & \cdots & 0 \\ 0 & \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_r & 0 & \cdots & 0 \\ 0 & 0 & \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_r & \cdots & 0 \\ & & & & \vdots & & & & \\ 0 & 0 & \cdots & 0 & \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_r \end{bmatrix} \cdot \begin{bmatrix} \|u^{(m)}\|_{p^*}^{p^*} \\ \|u^{(m)}\|_{2p^*}^{2p^*} \\ \|u^{(m)}\|_{3p^*}^{3p^*} \\ \vdots \\ \|u^{(m)}\|_{\ell p^*}^{\ell p^*} \end{bmatrix} =$$

$$\begin{bmatrix} \|u^{(m)}\|_{p^*}^{p^*} & \|u^{(m)}\|_{2p^*}^{2p^*} & \|u^{(m)}\|_{3p^*}^{3p^*} & \cdots & \|u^{(m)}\|_{(r+1)p^*}^{(r+1)p^*} \\ \|u^{(m)}\|_{2p^*}^{2p^*} & \|u^{(m)}\|_{3p^*}^{3p^*} & \|u^{(m)}\|_{4p^*}^{4p^*} & \cdots & \|u^{(m)}\|_{(r+2)p^*}^{(r+2)p^*} \\ \|u^{(m)}\|_{3p^*}^{3p^*} & \|u^{(m)}\|_{4p^*}^{4p^*} & \|u^{(m)}\|_{5p^*}^{5p^*} & \cdots & \|u^{(m)}\|_{(r+3)p^*}^{(r+3)p^*} \\ \vdots & \vdots & \vdots & & \vdots \\ \|u^{(m)}\|_{(\ell-r)p^*}^{(\ell-r)p^*} & \|u^{(m)}\|_{(\ell-r+1)p^*}^{(\ell-r+1)p^*} & \|u^{(m)}\|_{(\ell-r+2)p^*}^{(\ell-r+2)p^*} & \cdots & \|u^{(m)}\|_{\ell p^*}^{\ell p^*} \end{bmatrix} \cdot \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_r \end{bmatrix} = 0.$$

Therefore,

$$\begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_r \end{bmatrix} \in \text{null}\left(\begin{bmatrix} \|u^{(m)}\|_{p^*}^{p^*} & \|u^{(m)}\|_{2p^*}^{2p^*} & \|u^{(m)}\|_{3p^*}^{3p^*} & \cdots & \|u^{(m)}\|_{(r+1)p^*}^{(r+1)p^*} \\ \|u^{(m)}\|_{2p^*}^{2p^*} & \|u^{(m)}\|_{3p^*}^{3p^*} & \|u^{(m)}\|_{4p^*}^{4p^*} & \cdots & \|u^{(m)}\|_{(r+2)p^*}^{(r+2)p^*} \\ \|u^{(m)}\|_{3p^*}^{3p^*} & \|u^{(m)}\|_{4p^*}^{4p^*} & \|u^{(m)}\|_{5p^*}^{5p^*} & \cdots & \|u^{(m)}\|_{(r+3)p^*}^{(r+3)p^*} \\ \vdots & \vdots & \vdots & & \vdots \\ \|u^{(m)}\|_{(\ell-r)p^*}^{(\ell-r)p^*} & \|u^{(m)}\|_{(\ell-r+1)p^*}^{(\ell-r+1)p^*} & \|u^{(m)}\|_{(\ell-r+2)p^*}^{(\ell-r+2)p^*} & \cdots & \|u^{(m)}\|_{\ell p^*}^{\ell p^*} \end{bmatrix} \right).$$

Because the columns of

$$\begin{bmatrix} \alpha_1^{p^*} & \alpha_2^{p^*} & & \alpha_r^{p^*} \\ \alpha_1^{2p^*} & \alpha_2^{2p^*} & \cdots & \alpha_r^{2p^*} \\ \alpha_1^{3p^*} & \alpha_2^{3p^*} & & \alpha_r^{3p^*} \\ \vdots & \vdots & & \vdots \\ \alpha_1^{\ell p^*} & \alpha_2^{\ell p^*} & & \alpha_r^{\ell p^*} \end{bmatrix}$$

must be linearly independent when $\alpha_1, \alpha_2, \ldots$ are distinct (which is the case by the definition of our multiset formulation of the norm), then $r = \frac{\ell}{2}$ will determine a unique solution; thus the null space above is computed from a matrix with $r + 1$ columns and r rows, yielding a single vector for $(\gamma_0, \gamma_1, \ldots, \gamma_r)$. This vector can then be used to compute the roots of the polynomial $\gamma_0 + \gamma_1 x + \gamma_2 x^2 + \cdots + \gamma_r x^r$, which will determine the values $\{\alpha_1^{p^*}, \alpha_2^{p^*}, \ldots, \alpha_r^{p^*}\}$, which can each be taken to the $\frac{1}{p^*}$ power to compute $\{\alpha_1, \alpha_2, \ldots, \alpha_r\}$; the largest of those α_i values is used as the estimate of the maximum element in $u^{(m)}$. When $u^{(m)}$ contains at least r distinct values (i.e., $e_m \geq r$), then the problem will be well-defined; thus, if the roots of the null space spanning vector are not well-defined, then a smaller r can be used (and should be able to compute an exact estimate of the maximum, since $u^{(m)}$ can be projected exactly when r is the precise number of unique elements found in $u^{(m)}$).

To summarize, the null space projection method is performed as follows: First, standard FFT-based convolution is used for each p^* to compute the different norms at every index m. Then those norms are used to populate and compute the null space of a matrix. Finally the roots of the polynomial whose coefficients are given by the null space vector are computed to estimate the different $\alpha_1, \alpha_2, \ldots$. Note that this projection method is valid for any sequence of norms with even spacing: $\|u^{(m)}\|_{p_0+p^*}^{p_0^*+p^*}, \|u^{(m)}\|_{p_0+2p^*}^{p_0+2p^*}, \|u^{(m)}\|_{p_0+3p^*}^{p_0+3p^*}, \ldots \|u^{(m)}\|_{p_0+\ell p^*}^{p_0+\ell p^*}$.

The null space projection method can therefore be computed for an arbitrary r (i.e., it can be used to project to an arbitrary number of unique elements in $u^{(m)}$) by using linear algebra to compute the null space and to compute the roots of the polynomial (for instance, using the `numpy.roots` command in Python). For greater efficiency, setting r to a small constant allows us to precompute closed-form solutions of both the null space and the polynomial roots. For instance, using $r = 2$ (which is equivalent to projecting each vector $u^{(m)}$ to two unique values) results in a $\mathbb{R}^{3\times 2}$ null space computation and computing roots of a quadratic polynomial, both of which can be done in closed form **Algorithm 5**. Details of this $r = 2$ case can be found in **Appendix A**. In this case, it is possible to construct a series of powers of two with interleaved points, which guarantees that 4 evenly spaced p^* values exist (when the highest numerically stable p^* is higher than 2), meaning that the number of FFT calls is only twice what would be used by the original piecewise method (rather than four times the calls, which would be necessary if four evenly spaced points were placed at each considered p^* in a naive scheme). From algebraic and empirical evidence, we conjecture that the relative error of this $r = 2$ null space projection is bounded above by $1 - 0.7^{\frac{4}{p^*}}$, where p^* is the highest numerically stable p^* value.

This relative error is superior to the worst-case relative error when using a single p^*-norm estimate of the maximum. The relative error using the null space projection decreases rapidly as p^* is increased, meaning that the same procedure can be used to achieve an absolute error bound:

$$\hat{\alpha} - \|u^{(m)}\|_\infty < \tau^{\frac{1}{2p^*}} \left(1 - 0.7^{\frac{4}{p^*}}\right),$$

which achieves a unique maximum at

$$p^*_{\text{mode}} = \frac{1.4267 * \log(\tau) - 4.07094}{(\log(\tau) - 2.8534)\left(\log(1 - \frac{2.8534}{\log(\tau)})\right)} \approx 14.52.$$

Algorithm 5 Piecewise numerical max-convolution with projection, a numerical method to estimate the max-convolution of two PMFs or nonnegative vectors. This method uses a nullspace projection to achieve a closer estimate of the true maximum. Depending on the number of stable estimates, linear or quadratic projection is used. The parameters are two nonnegative vectors L' and R' (both scaled so that they have maximal element 1). The return value is a numerical estimate of the max-convolution $L' *_{\max} R'$.

```
 1: procedure NUMERICALMAXCONVOLVEPIECEWISEPROJECTIONAFFINE($L'$, $R'$, $p^*$)
 2:     $\ell_{\max} \leftarrow \text{argmax}_\ell\, L[\ell]$
 3:     $r_{\max} \leftarrow \text{argmax}_r\, R[r]$
 4:     $L' \leftarrow \frac{L}{L[\ell_{\max}]}$
 5:     $R' \leftarrow \frac{R}{R[r_{\max}]}$                    ▷ Scale to a proportional problem on $L'$,$R'$
 6:     allPStar $\leftarrow [2^{-1}, 2^0, 2^1, \ldots, 2 + 2^{\lfloor \log_2(p^*_{\max}) \rfloor}]$
 7:     for $h \in \{0, 1, \ldots, \texttt{len}(\text{allPStar})\}$ do
 8:         allPStarInterleaved[$2i$] $\leftarrow$ allPStar[$i$]
 9:         allPStarInterleaved[$2i+1$] $\leftarrow 0.5 \times$ (allPStar[$i$] + allPStar[$i+1$])
10:     end for
11:     for $i \in \{0, 1, \ldots, \texttt{len}(\text{allPStar})\}$ do
12:         resForAllPStar[$i$] $\leftarrow$ fftNonnegMaxConvolveGivenPStar($L'$, $R'$, allPStarInterleaved[$i$])
13:     end for
14:     for $m \in \{0, 1, \ldots, \texttt{len}(L) + \texttt{len}(R) - 1\}$ do
15:         maxStablePStarIndex[$m$] $\leftarrow \max\{i :\ (\text{resForAllPStar}[i][m])^{\text{allPStarInterleaved}[i]} \geq \tau)\}$
16:     end for
17:     for $o \in \{0, 1, \ldots, \texttt{len}(\text{maxStablePStarIndex})\}$ do
18:         maxStablePStarIndex[$o$] $-=$ maxStablePStarIndex[$o$] % 2      ▷ Restrict to powers of 2
19:     end for
20:     for $p \in \{0, 1, \ldots, \texttt{len}(\text{maxStablePStarIndex})\}$ do
21:         maxP $\leftarrow$ allPStarInterleaved[maxStablePStarIndex[$p$]]
22:         spacing $\leftarrow 0.25 \times$ maxP
23:         est$_4$ $\leftarrow$ resForAllPStar[maxStablePStarIndex[$p$]]
24:         est$_3$ $\leftarrow$ resForAllPStar[maxStablePStarIndex[$p$] $-$ 1]
25:         if maxStablePStarIndex[$p$] $< 5$ then   ▷ Need five $p^*$ in sequence to get four evenly spaced
26:             resForAllPStar[$p$] $\leftarrow$ maxLin(est$_3$, est$_4$)
27:         else
28:             est$_2$ $\leftarrow$ resForAllPStar[maxStablePStarIndex[$p$] $-$ 2]
29:             est$_1$ $\leftarrow$ resForAllPStar[maxStablePStarIndex[$p$] $-$ 4]  ▷ Index - 4 is the next evenly spaced
    point
30:             resForAllPStar[$p$] $\leftarrow$ maxQuad(est$_1$, est$_2$, est$_3$, est$_4$, spacing)
31:         end if
32:     end for
33:     result $\leftarrow$ affineCorrect(resForAllPStar, maxStablePStarIndex)
34:     return $L[\ell_{\max}] \times R[r_{\max}] \times$ result                ▷ Undo previous scaling
35: end procedure
```

As before, the worst-case absolute error of the unscaled problem will be found by simply scaling the absolute error at p^*_{mode}:

$$\max_{\ell} L[\ell] \, \max_{r} R[r] \, \tau^{\frac{1}{2p^*_{\text{mode}}}} \left(1 - 0.7^{\frac{4}{p^*_{\text{mode}}}}\right).$$

Because p^*_{mode} (the value of p^* producing the worst-case absolute error) for the null space projection method it is invariant to the length of the list k (enabling us to compute a numeric value), and because its numeric value is so small, even a fairly small choice of $p^*_{\max}$ will suffice (now $p^*_{\max} \in O(1)$ rather than in $O(\log(k))$ as it was with the original piecewise method).

The one caveat of this worst-case absolute error bound is that it presumes at least four evenly spaced, stable p^* can be found (which may not be the case by choosing p^* from the sequence 2^i in cases when $\|u^{(m)}\|_\infty \approx 0$); however, assuming standard fast convolution can be performed (a reasonable assumption given it is one of the essential numeric algorithms), then four evenly spaced p^* values could be chosen very close to 1; therefore, these values of p^* could be added to the sequence so that the algorithm is slightly slower, but essentially always yield this worst-case absolute error bound.

In practice, we can demonstrate that the null space projection method is very accurate. First we show the impact of using the quadratic (i.e., $r = 2$) projection method on unscaled single $u^{(m)}$ vectors. The projection method was tested on vectors of different lengths drawn from different types of Beta distributions and are compared with the results of the p-norms with the highest stable p (**Figure 3.5**). The relative errors between the original piecewise method and the null space projection method are compared using a max-convolution on two randomly created input PMFs of length 1024 (**Figure 3.6**). Note that the null space projection can also be paired with affine scaling on the back end, just as the original piecewise method can be. In practice, the null space projection increases the accuracy demonstrably on a variety of different problems, although the original piecewise method also performs well.

Although the worst-case runtime of the null space projection method is roughly twice that of the original piecewise method, the error bound no longer depends on the length of the result k. Thus, for a given relative error bound on the top contour (i.e., the equivalent of the derivation of $p^*_{\max}$ in the original piecewise algorithm), the value of $p^*_{\max}$ is fixed and no longer in $O(\log(k))$. For example, achieving a 0.5% relative error in the top contour would require

$$1 - 0.7^{\frac{4}{p^*_{\max}}} \leq 0.005 \rightarrow p^*_{\max} \geq 4\frac{\log(0.7)}{\log(0.995)} \approx 284.62,$$

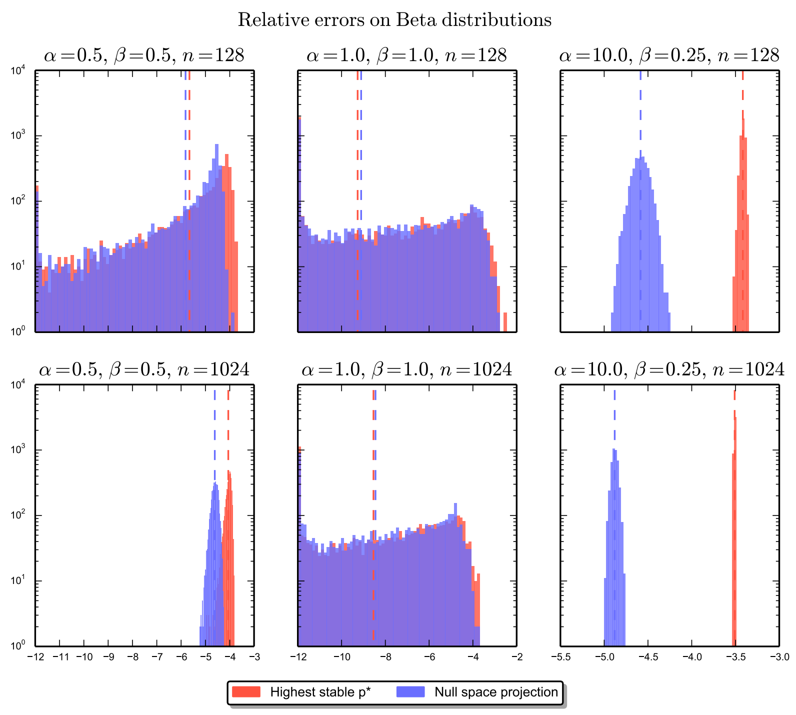


Figure 3.5: Relative errors on random vectors with and without null space projection. For the two approximation methods (using the highest stable p^*-norm with the heuristically chosen $p^*_{\max}$ or using the null space projection with $p^*_{\max} = 64$), vectors of different lengths are sampled (2^{12} repetitions) from a variety of Beta distributions. The settings for the parameters (α, β) of the Beta distribution that were used, as well as the lengths of the generated vectors are shown in the titles of the subplots: $\alpha = 0.5, \beta = 0.5$ (bimodal with modes near zero and one); $\alpha = 0.1, \beta = 0.1$ (uniform distribution); $\alpha = 10, \beta = 0.25$ (with a strong mode near one). The red area depicts the frequencies (y-axis) of the different magnitudes of (relative) error (x-axis) when using the highest stable p^*-norm is used as an approximation of the Chebyshev norm ($p = \infty$). The blue area shows the errors with the method that performs a projection (either quadratic or linear depending on how many numerically stable p^* are available) to estimate the Chebyshev norm.

meaning that choosing $p^*_{\max} = 512$ would achieve a very high accuracy, but while only performing 2×9 FFTs. For very large vectors, this will not be substantially more expensive than the original piecewise algorithm, which uses a higher value of $p^*_{\max}$ (in this case, $p^*_{\max} = \log_{1.005}(k)$, which continues to grow as k does) to keep the error lower in practice. As a result, the runtime of the null space projection approximation is in $O(k \log(k))$ rather than $O(k \log(k) \log(\log(k)))$, despite the similar runtime in practice to the original piecewise method (the null space projection method uses twice as many FFTs performed per p^* value, but requires slightly fewer p^* values).

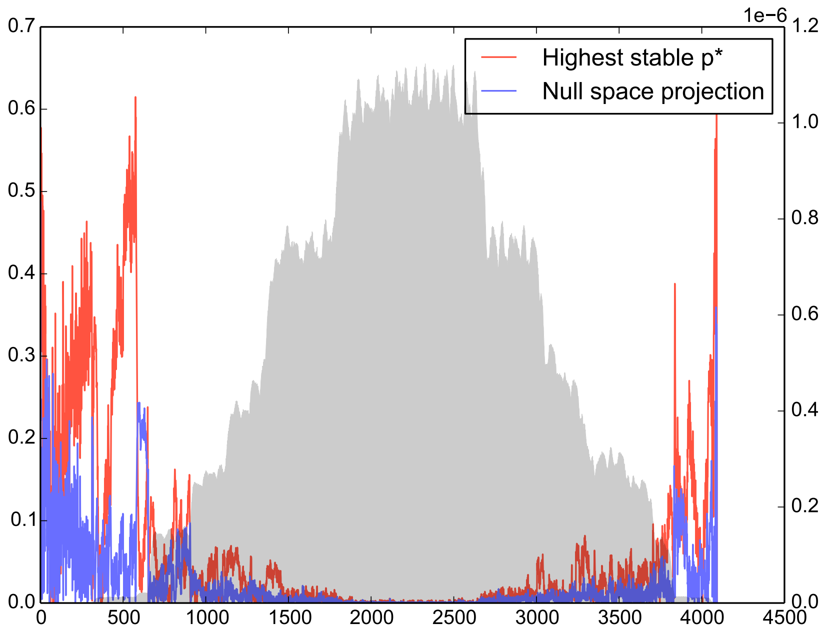


Figure 3.6: Relative errors on large max-convolution with and without null space projection. Max-convolution between two randomly generated vectors (both uniform vectors convolved with narrow Gaussians with uniform noise added afterward), performed with the highest stable p^*-norm (using the heuristic choice of p^*_{max} for problems of this size) and with null space projection (using $p^*_{\text{max}} = 64$). The left y-axis shows the relative error at index m. Associated with that, you can see the red and blue curve depicting the errors from the two different methods: Red describes the max-norm estimation using only the highest stable p^* while purple was generated using quadratic projection at the four highest stable p^* values (when at least four evenly spaced values are numerically stable) and linear projection at the two highest stable p^* values (when only two p^* are numerically stable). The results of both approaches are corrected with the affine transformation method proposed in this manuscript. In the background the gray shaded curve shows the exact result of the max-convolution at every index (to be used with the second y-axis on the right).

k	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
Naive	**0.0142**	0.0530	0.192	0.767	3.03	12.1	48.2
Naive (vectorized)	0.0175	0.0381	0.0908	0.251	0.790	2.75	10.1
FILL1 ([14])	0.0866	1.09	7.21	19.4	457	—	—
Max. stable p^*, affine corr.	0.0277	0.0353	0.0533	0.0848	0.149	0.274	0.537
Projection, affine corr.	0.0236	**0.0307**	**0.0467**	**0.0760**	**0.137**	**0.258**	**0.520**

Table 3.1: Runtimes of different methods for max-convolution on uniform vectors of length k. The runtimes were gathered using the `timeit` package in Python. They include all preprocessing steps necessary for the algorithm (e.g., sorting prior to the FILL1 approach). The values are total runtimes (in seconds) to run 5 repetitions on different, randomly generated vectors. FILL1 was not run on larger problems, because it ran substantially longer than the non-vectorized naive approach. On the two approximation methods presented in this manuscript, the highest stable p^*-norm approximation was run with the heuristically chosen $p^*_{\max}$ for problems of the appropriate size and the null space projection was run with $p^*_{\max} = 64$.

3.3.5 Practical runtime comparison

To compare the actual runtimes of the final algorithm developed in this manuscript with a naive max-convolution and a previously proposed method from Bussieck et al. [14], all methods were run on vectors of different random (uniform in $[0, 1]$) composition and length (k). The first and second input vector were generated seperately but are always of same length. **Table 3.1** shows the result of this experiment. All methods were implemented in Python, using `numpy` where applicable (e.g., to vectorize). A non-vectorized version of naive max-convolution was included to estimate the effects of vectorization. The approach from Bussieck et al. ran as a reimplementation based on the pseudocode in their manuscript. From their variants of proposed methods, FILL1 was chosen because of its use in their corresponding benchmark and its recommendation by the authors for having a lower runtime constant in practice compared to other methods they proposed. The method is based on sorting the input vectors and traversing the (implicitly) resulting partially ordered matrix of products in a way that not all entries need to be evaluated, while only keeping track of the so-called cover of maximal elements. FILL1 already includes some more sophisticated checks to keep the cover small and thereby reducing the overhead per iteration. Unfortunately, although we observed that the FILL1 method requires between $O(n \log(n))$ and $O(n^2)$ iterations in practice, this per-iteration overhead results in a worst-case cost of $\log(n)$ per iteration, yielding an overall runtime in practice between $O(n \log(n) \log(n))$ and $O(n^2 \log(n))$. As the authors state, this overhead is due to the expense of storing the cover, which can be implemented e.g., using a binary heap (recommended by the authors and used in this reimplementation). Additionally, due to the fairly sophisticated datastructures needed for this algorithm it had a higher runtime constant than the other methods presented here, and furthermore we saw no means to vectorize it to improve the efficiency. For this reason, it is not truly fair to compare the raw runtimes to the other vectorized algorithms (and it is not likely that this Python reimplementation is as efficient as the original version, which[14] implemented in ANSI-C); however, comparing a non-vectorized implementation of the naive $O(n^2)$ approach with its vectorized counterpart gives an estimated $\approx 5\times$ speedup from vectorization, suggesting that it is not substantially faster than the naive approach on these problems (it should be noted that whereas the methods presented here have tight runtime bounds but produce approximate results, the FILL1 algorithm is exact, but its runtime depends on the data processed). During investigation of these runtimes, we found that on the given problems, the proposed average case of $O(n \log(n))$ iterations was rarely reached. A reason might be an unrecognized violation of the assumptions of the theory behind this theoretical average runtime in how the input vectors were generated.

In contrast to the exact method from Bussieck et al. [14], the herein proposed approximate procedure are faster whenever the input vectors are at least 128 elements long (shorter vectors are most efficiently processed with the naive approach). The null space projection method is the fastest method presented here (because it can use a lower $p^*_{\max}$), although the higher density of p^* values it uses (and thus, additional FFTs) make the runtimes nearly identical for both approximation methods.

3.3.6 Demonstration on hidden Markov model with Toeplitz transition matrix

One example that profits from fast max-convolution of non-negative vectors is computing the Viterbi path using a hidden Markov model (HMM) (i.e., the *maximum a posteriori* states) with an additive transition function satisfying $\Pr(X_{i+1} = a | X_i = b) \propto \delta(a - b)$ for some arbitrary function δ (δ can be represented as a table, because we are considering all possible discrete functions). This additivity constraint is equivalent to the transition matrix being a “Toeplitz matrix”: the transition matrix $T_{a,b} = \Pr(X_{i+1} = a | X_i = b)$ is a Toeplitz matrix when all cells diagonal from each other (to the upper left and lower right) have identical values (i.e., $\forall a, \forall b,\ T_{a,b} = T_{a+1,b+1}$). Because of the Markov property of the chain, we only need to max-marginalize out the latent variable at time i to compute the distribution for the next latent variable X_{i+1} and all observed values of the data variables $D_0 \ldots D_{i+1}$. This procedure, called the Viterbi algorithm, is continued inductively:

$$\max_{x_0,x_1,\ldots x_{i-1}} \Pr(D_0,D_1,\ldots D_{i-1},X_0=x_0,X_1=x_1,\ldots,X_i=x_i)=$$

$$\max_{x_{i-1}} \max_{x_0,x_1,\ldots x_{i-2}} \Pr(D_0,D_1,\ldots D_{i-2},X_0=x_0,X_1=x_1,\ldots,X_{i-1}=x_{i-1})$$
$$\Pr(D_{i-1}|X_{i-1}=x_{i-1})\Pr(X_i=x_i|X_{i-1}=x_{i-1})$$

and continuing by exploiting the self-similarity on a smaller problem to proceed inductively with the vector variable “fromLeft”, revealing a max-convolution (for this specialized HMM with additive transitions):

$$\max_{x_0,x_1,\ldots x_{i-1}} \Pr(D_0,D_1,\ldots D_{i-1},X_0=x_0,X_1=x_1,\ldots,X_i=x_i)=$$
$$\max_{x_{i-1}} \text{fromLeft}[i-1]\Pr(D_{i-1}|X_{i-1}=x_{i-1})\delta[x_i-x_{i-1}]=$$
$$(\text{fromLeft}[i-1]\ \text{likelihood}[D_{i-1}]) *_{\max} \delta[x_i-x_{i-1}].$$

After computing this left-to-right pass (which consisted of $n-1$ max-convolutions and vector multiplications), we can find the *maximum a posteriori* configuration of the latent variables $X_0,\ldots X_{n-1}=x_0^*,\ldots x_{n-1}^*$ backtracking right-to-left, which can be done by finding the variable value x_i that maximizes fromLeft$[i][x_i]\times\delta[x_{i+1}^*-x_i]$ (thus defining x_i^* and enabling induction on the right-to-left pass). The right-to-left pass thus requires $O(nk)$ steps (**Algorithm 6**). Note that the full max-marginal distributions on each latent variable X_i can be computed via a small modification, which would perform a more complex right-to-left pass that is nearly identical to the left-to-right pass, but which performs subtraction instead of addition (i.e., by reversing the vector representation of the PMF of the subtracted argument before it is max-convolved;[141]).

We apply this HMM with additive transition probabilities to a data analysis problem from economics. It is known for example, that the current figures of unemployment in a country have (among others) impact on prices of commodities like oil. If one could predict unemployment figures before the usual weekly or monthly release by the responsible government bureaus, this would lead to an information advantage and an opportunity for short-term arbitrage. The close relation of economic indicators like market prices and stock market indices (especially of indices combining several stocks of different industries) to unemployment statistics can be used to tackle this problem.

In the following demonstration of our method, we create a simple HMM with additive transitions and use it to infer the *maximum a posteriori* unemployment statistics

Algorithm 6 Viterbi for models with additive transitions, which accepts the length k vector *prior*, a list of n binned observations *data*, a $a \times k$ matrix of likelihoods (where a is the number of bins used to discretize the data) *likelihoods*, and a length $2k-1$ vector δ that describes the transition probabilities. The algorithm returns a Viterbi path of length n, where each element in the path is a valid state $\in \{0, 1, \dots k-1\}$.

```
procedure ViterbiForAdditiveTransitions(prior, data, likelihood, $\delta$)
    fromLeft[0] ← prior
    for $i = 0$ to $n-2$ do
        fromLeft[$i$] ← fromLeft[$i$] × likelihood[data[$i$]]
        fromLeft[$i+1$] ← fromLeft[$i$] $*_{\max}$ $\delta$
    end for
    fromLeft[$n$] ← fromLeft[$n$] × likelihood[data[$i$]][$n$]]

    path[$n-1$] ← $\text{argmax}_j$ fromLeft[$n-1$][$j$]
    for $i = n-2$ to 0 do
        maxProdPosterior ← −1
        argmaxProdPosterior ← −1
        for $l = k$ to 1 do
            currProdPosterior ← fromLeft[$i$] × $\delta$[$l$ − path[$i+1$]]
            if currProdPosterior > maxProdPosterior then
                maxProdPosterior ← currProdPosterior
                argmaxProdPosterior ← $l$
            end if
        end for
        path[$i$] ← argmaxProdPosterior
    end for
    return path
end procedure
```

given past history (i.e., how often unemployment is low and high, as well as how often unemployment goes down or up in a short amount of time) and current stock market prices (the observed data). We discretized random variables for the observed data (S&P 500, adjusted closing prices ; retrieved from YAHOO! historical stock prices: `http://data.bls.gov/cgi-bin/surveymost?blsseriesCUUR0000SA0`), and "latent" variables (unemployment insurance claims, seasonally adjusted, were retrieved from the U.S. Department of Labor: `https://www.oui.doleta.gov/unemploy/claims.asp`). Stock prices were additionally inflation adjusted by (i.e., divided by) the consumer price index (CPI) (retrieved from the U.S. Bureau of Labor Statistics: `https://finance.yahoo.com/q?s=^GSPC`). The intersection of both "latent" and observed data was available weekly from week 4 in 1967 to week 52 in 2014, resulting in 2500 data points for each type of variable.

To investigate the influence of overfitting, we partition the data in two parts, before June 2005 and after June 2005, so that we are effectively training on $\frac{2000\times 100}{2500} = 80\%$ of the data points, and then demonstrate the Viterbi path on the entirety of the data (both the 80% training data and the 20% of the data withheld from empirical parameter estimation). Unemployment insurance claims were discretized into 512 and stock prices were discretized into 128 bins. Simple empirical models of the prior distribution for unemployment, the likelihood of unemployment given stock prices, and the transition probability of unemployment were built as follows: The initial or prior distribution for unemployment claims at $i = 0$ was calculated by marginalizing the time series of training data for the claims (i.e., counting the number of times any particular unemployment value was reached over all possible bins). Our transition function (the conditional probability $\Pr(X_{i+1}|X_i)$) similarly counts the number of times each possible change $X_{i+1} - X_i \in \{-511, -510, \ldots 511\}$ occurred over all available time points. Interestingly, the resulting transition distribution roughly resembles a Gaussian (but is not an exact Gaussian). This underscores a great quality of working with discrete distributions: while continuous distributions may have closed-forms for max-convolution (which can be computed quickly), discrete distributions have the distinct advantage that they can accurately approximate any smooth distribution. Lastly, the likelihoods of observing a stock price given the unemployment at the same time were trained using an empirical discrete joint distribution based on the historical data.

We compute the Viterbi path two times: First we use naive, exact max-convolution, which requires a total of $O(nk^2)$ steps. Second, we use fast numerical max-convolution, which requires $O(n\, k \log(k) \log(\log(k))$ steps. Despite the simplicity of the model, the exact Viterbi path (computed via exact max-convolution) is highly informative for predicting the value of unemployment, even for the 20% of the data that were not

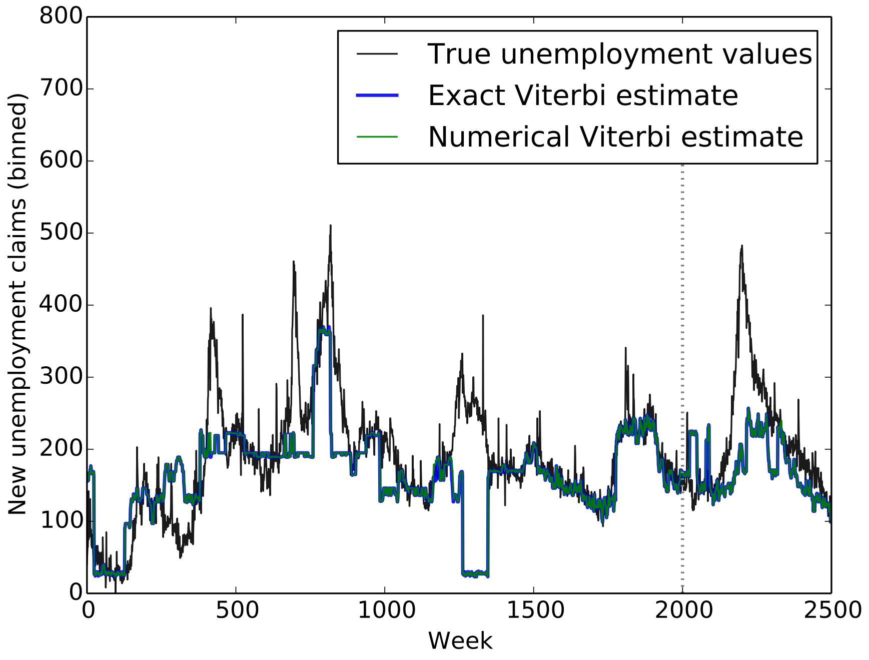


Figure 3.7: Viterbi analysis of employment given stock index values. The Viterbi path corresponding to the *maximum a posteriori* prediction of the number of new unemployment insurance claims is produced for a model where the state transition probabilities are additive. The exact Viterbi estimate tracks well with the true unemployment values. Training parameters were taken from only the true unemployment data to the left of the vertical dotted line; however, the Viterbi paths to the right of the dotted line (where unemployment data were withheld from the likelihood, prior, and transition parameters) also track well with the true unemployment statistics. The Viterbi path computed with fast numerical max-convolution (via the null space projection piecewise approach) is nearly identical to the result computed with the slower exact approach. Note that for this problem, the original, simpler piecewise approach also produces very similar results.

used to estimate the empirical prior, likelihood, and transition distributions. Also, the numerical max-convolution method is nearly identical to the exact max-convolution method at every index (**Figure 3.7**). Even with a fairly rough discretization (i.e., $k = 512$), the fast numerical method (via the original, simpler piecewise algorithm with $p^*_{\max} = 8192$) used 141.4 seconds compared to the 292.3 seconds required by the naive approach. The higher-precision projection algorithm (which uses a smaller $p^*_{\max}$, but calls FFT twice for each power of two p^*) computes nearly identical result using $p^*_{\max} = 64$ for this problem in 136.6 seconds. The speedup of both fast numerical algorithms relative to the naive quadratic max-convolution algorithm will increase dramatically as k is increased.

3.4 Discussion

Both piecewise numerical max-convolution methods are highly accurate in practice and achieve a substantial speedup over both the naive approach and the approach proposed by Bussieck et al.[14]. This is particularly true for large problems: For the original piecewise method presented here, the $\log_2(\log_{1+\tau^{\frac{1}{4}}}(k))$ multiplier may never be small, but it grows so slowly with k that it will be < 18 even when k is on the same order of magnitude as the number of particles in the observable universe. This means that, for all practical purposes, the method behaves asymptotically as a slightly slower $O(k \log_2(k))$ method, which means the speedup relative to the naive method becomes more pronounced as k becomes large. For the second method presented (the null space projection), the runtime for a given relative error bound will be in $O(k \log_2(k))$. In practice, both methods have similar runtime on moderate or large problems.

The basic motivation of the first approach described—i.e., the idea of approximating the Chebyshev norm with the largest p^*-norm that can be computed accurately, and then convolving according to this norm using FFT—also suggests further possible avenues of research. For instance, it may be possible to compute a single FFT (rather than an FFT at each of several contours) on a more precise implementation of complex numbers. Such an implementation of complex values could store not only the real and imaginary components, but also other much smaller real and imaginary components that have been accumulated through $+$ operations, even those which have small enough magnitudes that they are dwarfed by other summands. With such an approach it would be possible to numerically approximate the max-convolution result in the same overall runtime as long as only a bounded “history” of such summands was recorded (i.e., if the top few magnitude summands—whether that be the top 7 or the top $\log_2(\log_{1+\tau^{\frac{1}{4}}}(k))$—was stored and operated on). In a similar vein, it would be interesting to investigate the utility of complex values that use rational numbers (rather than fixed-precision floating point values), which will be highly precise, but will increase in precision (and therefore, computational complexity of each arithmetic operation) as the dynamic range between the smallest and largest nonzero values in L and R increases (because taking L' to a large power p^* may produce a very small value). Other simpler improvements could include optimizing the error vs. runtime trade-off between the log-base of the contour search: the method currently searches $\log_2(p^*_{\max})$ contours, but a smaller or larger log-base could be used in order to optimize the trade-off between error and runtime.

It is likely that the best trade-off will occur by performing the fast p^*-norm convolution with a number type that sums values over vast dynamic ranges by appending them in a short (i.e., bounded or constant size) list or tree and sums values within the same dynamic range by querying the list or tree and then summing in at the appropriate magnitude. This is reminiscent of the fast multipole algorithm[128]. This would permit the method to use a single large p^* rather than a piecewise approach, by moving the complexity into operations on a single number rather than by performing multiple FFTs with simple floating-point numbers.

The basic motivation of the second approach described—i.e., using the *sequence* of p^*-norms (each computed via FFT) to estimate the maximum value—generalizes the p^*-norm fast convolution numerical approach into an interesting theoretical problem in its own right: given an oracle that delivers a small number of norms (the number of norms retrieved must be $c \in o(k)$ to significantly outperform the naive quadratic approach) about each vector $u^{(m)}$, amalgamate these norms in an efficient manner to estimate the maximum value in each $u^{(m)}$. This method may be applicable to other problems, such as databases where the maximum values of some combinatorial operation (in this case the *maximum a posteriori* distribution of the sum of two random variables $X+Y$) is desired but where caching all possible queries and their maxima would be time or space prohibitive. In a manner reminiscent of how we employ FFT, it may be possible to retrieve moments of the result of some combinatoric combination between distributions on the fly, and then use these moments to approximate true maximum (or, in general, other sought quantities describing the distribution of interest).

In practice, the worst-case relative error of our quadratic approximation is quite low. For example, when $p^* = 8$ is stable, then the relative error is less than 2.3%, regardless of the lengths of the vectors being max-convolved. In contrast, the worst-case relative error using the original piecewise method would be $\leq k^{\frac{1}{16}} - 1$, where k is the length of the max-convolution result (when $n = 1024$, the relative error of the original piecewise method would be $\approx 54\%$).

Of course, the use of the null space projection method is predicated on the existence of at least four sequential p^* points, but it would be possible to use finer spacing between p^* values (e.g., $p^* \in (1, 1.01, 1.02, 1.03)$ to guarantee that this will essentially be the case as long as FFT (i.e., $p^* = 1$) is stable. But more generally, the problem of estimating extrema from p^*-norms (or, equivalently, from the p^*-th roots of the p^*-th moments of a distribution with bounded support), will undoubtedly permit many more possible approaches that we have not yet considered. One that would be compelling is to relate the Fourier transform of the sequential moments to the maximum value in the distribution; such an approach could permit all stable p^* at any index m to be used to efficiently approximate the maximum value (by computing the FFT of the sequence of norms). Such new adaptations of the method could permit low worst-case error without any noticable runtime increase.

3.4.1 Multidimensional numerical max-convolution

The fast numerical piecewise method for max-convolution (and the affine piecewise modification) are both applicable to matrices as well as vectors (and, most generally, to tensors of any dimension). This is because the p^*-norm (as well as the derived error bounds as an approximation of the Chebyshev norm) can likewise approximate the maximum element in the tensor $u^{(m_1,m_2,\ldots)}$ generated to find the max-convolution result at index $m_1, m_2, \ldots$ of a multidimensional problem, because the sum

$$\sum_{i_1,i_2,\ldots} \left(u^{(m_1,m_2,\ldots)}_{i_1,i_2,\ldots}\right)^{p^*}$$

computed by convolution corresponds to the Frobenius norm (i.e., the "entrywise norm") of the tensor, and after taking the result of the sum to the power $\frac{1}{p^*}$, will converge to the maximum value in the tensor (if p^* is large enough).

This means that the fast numerical approximation, including the affine piecewise modification, can be used without modification by invoking standard multidimensional convolution (i.e., $*$). Matrix (and, in general, tensor) convolution is likewise possible for any dimension via the row-column algorithm, which transforms the FFT of a matrix into sequential FFTs on each row and column. The accompanying Python code demonstrates the fast numerical max-convolution method on matrices, and the code can be run on tensors of any dimension (without requiring any modification).

The speedup of FFT tensor convolution (relative to naive convolution) becomes considerably higher as the dimension of the tensors increases; for this reason, the speedup of fast numerical max-convolution becomes even more pronounced as the dimension increases. For a tensor of dimension d and width k (i.e., where the index bounds of every dimension are $\in \{0, 1, \ldots k-1\}$), the cost of naive max-convolution will be in $O(k^{2d})$, whereas the cost of numerical max-convolution is $O(k^d \log_2(k))$ (ignoring the $\log_2(\log_{1+\tau^{\frac{1}{4}}}(k)) \leq 18$ multiplier), meaning that there is an $O(\frac{k^d}{d \log_2(k)})$ speedup from the numerical approach. Examples of such tensor problems include graph theory, where adjacency matrix representation can be used to describe respective distances between nodes in a network.

As a concrete example, the Python demo code computes the max-convolution between two 256×256 matrices. The naive method required 494 seconds, but the numerical result with the original piecewise method was computed in 3.18 seconds (yielding a maximum absolute error of 0.0173 and a maximum relative error of 0.0511) and the numerical result with the null space projection method was computed in 3.99 seconds (using $p^*_{\max} = 512$, which corresponds to a relative error of $< 0.1\%$ in the top contour, yielding a maximum absolute error of 0.0141 and a maximum relative error of 0.0227) and in 3.05 seconds (using $p^*_{\max} = 64$, which corresponds to a relative error of $< 2.5\%$ in the top contour, yielding a maximum absolute error of 0.0667 and a maximum relative error of 0.067). Not only does the speedup of the proposed methods relative to naive max-convolution increase significantly as the dimension of the tensor is increased, no other faster-than-naive algorithms exist for max-convolution of matrices or tensors.

Multidimensional max-convolution can likewise be applied to hidden Markov models with additive transitions over multidimensional variables (e.g., allowing the latent variable to be a two-dimensional joint distribution of American and German unemployment with a two-dimensional joint transition probability).

3.4.2 Max-deconvolution

The same p^*-norm approximation can also be applied to the max-deconvolution problem (i.e., solving $M = L *_{\max} R$ for R when given M and L). This can be accomplished by computing the ratio of FFT(M^{p^*}) to FFT(L^{p^*}) (assuming L has already been properly zero-padded), and then computing the inverse FFT of the result to approximate R^{p^*}; however, it should be noted that deconvolution methods are typically less stable than the corresponding convolution methods, computing a ratio is less stable than computing a product (particularly when the denominator is close to zero).

Chapter 4

Protein inference with loopy belief propagation on Bayesian networks

This chapter revolves around the combination of the algorithmic improvements from the last chapter with an approximate inference method that can utilize convolution trees efficiently to answer queries on a refined state-of-the-art model for protein inference into a ready-to-use tool working on OpenMS' data structures. It shows results from benchmarks for accuracy on a ground truth data set and efficiency on a large-scale human sample measured on a long gradient.

4.1 Introduction

Although the benefits of probabilistically dividing evidence from shared peptides are long known[108], the high complexity of current Bayesian inference methods[82,83,135] for accurate posteriors on the full protein-peptide graph makes a rigorous probabilistic handling unsuitable for large data sets (as described in detail in Background Section 2.4.5). In our new approach EPIFANY (Efficient Protein InFerence for ANY protein-peptide network) we use a fast approximate inference algorithm called loopy belief propagation (LBP) which has already been shown to perform well on solving other types of probabilistic graphical models (e.g., models used in important information theoretic algorithms like the error-correcting turbo codes[10] as well as on quick medical reference (QMR) disease diagnosis networks[99]). Using LBP we can achieve drastically improved runtimes than other Bayesian approaches without any approximations on the underlying graph itself. Additionally, we improved the calibration of the resulting FDRs by introducing an optional regularized model with max-product inference and a greedy protein group resolution based on the reported protein probabilities.

Partly reprinted and adapted to fit the style and flow of this thesis with permission from:

EPIFANY - A method for efficient high-confidence protein inference

J. Pfeuffer, T. Sachsenberg, T. M. Dijkstra, O. Serang, K. Reinert, and O. Kohlbacher

Journal of Proteome Research p. 734327, (2019), Copyright 2019 American Chemical Society.

Author	Author position	Scientific ideas %	Data generation %	Analysis & interpretation %	Paper writing %
J. Pfeuffer	1	40	95	80	90
T. Sachsenberg	2	5	5	5	5
T. Dijkstra	3	5	0	5	1
O. Serang	4	25	0	0	1
K. Reinert	5	10	0	5	1
O. Kohlbacher	6	15	0	5	2
Status:	published				

4.2 Methods

In the following section we define the underlying probabilistic graphical model used to describe the protein inference problem including all its conditional dependencies. Then we present the inference algorithm, which allows to efficiently calculate posterior probabilities for peptides, proteins and protein groups. In addition, we provide information on the pre- and post-processing of the data used to present the results later on.

4.2.1 Model

The model we chose for protein inference is based on a Bayesian network (BN) representation. The protein-peptide graph (Figure 2.7) encodes the conditional dependencies of proteins and their peptides. The advantages of this specification of conditional (in-)dependencies is the resulting factorization of the high-dimensional joint distribution into smaller distributions, namely prior distributions for the proteins and conditional probability distributions (CPDs) for the peptides given their parents. In case of the binary representation for every peptide and protein, these distributions are discrete and correspond to conditional probability tables (CPTs).

Additionally, the Bayesian network needs to be parameterized. Although the factorization into smaller distributions decreases the number of parameters, each CPT still needs 2^p parameters, where p is the number of parent nodes, to be set or learned. By recognizing the fact that in the generative process from proteins to peptides the presence of any of the parent proteins is enough to potentially produce a peptide, we can reduce the number of parameters further when specifying the conditional probability according to a noisy-OR model[113]. In its original form, the network using the noisy-OR model requires the following parameters:

- γ_ρ prior for protein with index ρ
- $\alpha_{\rho,\varepsilon}$ noisy-OR emission probability of a protein ρ generating peptide ε
- β_ε noisy-OR leak probability for a peptide ε being generated by chance

For now, we employ the same simplification used in the Fido[144] algorithm by assuming equality among all α and the presence of only one β. Additionally, a constant prior γ for all proteins prevents biases on protein level when no further information is available. The parameters α, β, γ either have to be specified manually, or are by default selected from a grid of initial values based on target-decoy classification performance and probability calibration (see Subsection 4.2.3 for implementation details).

One addition to the original model is an option to add prior probabilities for the random variables that model the number of proteins which might produce a certain shared peptide. These priors were designed empirically to decrease with a rising number of present parents. In the course of inference, this results in a form of regularization on the number of parent proteins and a more uneven distribution of the evidence from shared peptides starting at the most likely producing proteins (based on their beliefs from the rest of the network), especially in conjunction with so-called max-product inference. For a more detailed description of the parameterization including equations and an example please refer to the Appendix B.1.1 and B.1.2.

4.2.2 Algorithm

Once the protein inference problem is modeled as a (factorized) probability distribution, probabilities can be inferred on random variables of interest. Usually, probabilities of interest are the marginal probabilities of proteins, protein groups or peptides, given the evidence on the PSM level, the so-called posterior probabilities.

As peptide level evidence, the algorithm first reads the PSM probabilities and the associations to their parent proteins from spectra searched with a peptide search engine. It then aggregates PSM probabilities on the peptide level by picking the maximum PSM probability per unmodified peptide sequence and filters out peptides with extremely small probabilities (e.g., below 0.001).

A naive approach to inference in a Bayesian network would be to create a large joint probability table for each connected component of the network and marginalize for each protein by aggregating the probabilities of all possible configurations leading to the same state of the current protein of interest. To make this approach viable for at least small to medium-sized problems, previous tools resorted to (Gibbs) sampling[84] or sped up calculations by caching results and making use of symmetries arising due to the chosen model[144]. A possible symmetry to be exploited is the dependence of the peptide probability only on the number of parent proteins (not their exact combination). This symmetry consideration reduces the number of different input configurations in the presence of indistinguishable groups. Although this procedure has been implemented efficiently in tools like Fido, the worst-case runtime is exponential in the number of proteins in a connected component. Reducing the size of the connected components by splitting them at low-probability peptides is a reasonable approximation only if the probability cutoff is not too high. Using the looping version[99] of Pearl's belief propagation algorithm[112], even non-tree structured graphs with cycles (such as all but the simplest protein-peptide networks) can be processed efficiently while keeping flexibility in which types of factors (probability table-based factors, function-like factors, convolution-tree-based adder factors, etc.) on which sets of random variables are used. Convolution trees[141] (CTs) leverage fast Fourier transformation to calculate the convolution of probability distributions. Additionally, convolutions are performed in a hierarchical procedure to keep the sizes of intermediate tables as small as possible. EPIFANY uses CTs to propagate probabilities between peptide and protein (group) level efficiently even at peptides with many parents (see Appendix B.1.2 for details). To apply loopy belief propagation to our problem we create a factor graph (see Figure 4.1 and Subsection 4.2.3) from our model and initialize messages on all edges in both directions uniformly. In the following we will use the term "belief" for the current marginal probabilities of a set of variables in a factor. All beliefs on a factor are initialized according to their priors, evidence and/or the likelihoods calculated based on the parameterization (i.e., noisy-"OR" parameters and regularization). Lastly, starting from factors that carry initial information, the algorithm iteratively queues, updates and passes messages between the factors (to update their beliefs) until messages do not change anymore (i.e., convergence is reached in terms of their mean-squared error).

By default, messages with the highest residuals compared to the last message, gain highest priority for the next iteration[38]. Beliefs are updated by incoming messages following the HUGIN algorithm, which means that for example additional beliefs on the so-called separator variables are cached on the graph's edges[62]. Details about the algorithm can be found in the Appendix on an applied example (Appendix B.1.2), or in a more general setting in the Background (Section 2.4.5). Since loopy belief propagation is an approximate algorithm on non-tree structured parts of the network and convergence is an important concern, messages can additionally be dampened (by a “momentum”) in a way that the updated message is a convex combination of old and new messages[99]. The idea of damping for solving the problem of oscillating messages is further discussed in the Limitations Section 4.3.5 and on an example in the Appendix B.1.3.

4.2.3 Implementation

EPIFANY starts with the output from probabilistic rescoring tools applied to peptide search engine results. It writes the PSM probabilities and associations from PSMs to proteins into OpenMS' datastructures. After aggregating PSM probabilities on the peptide level and creating the bipartite graph from the peptides mapping to potential protein candidates as described in the previous section, the resulting protein-peptide graph is split into connected components by a depth-first search. Then—using the OpenMP 2.0 API[110] in a dynamic scheduling mode—the processing of the connected components is distributed on as many CPU threads as allowed by a user-specified parameter. For each connected component in the graph a factor graph is built, equivalent to the Bayesian network specified in Section 4.2.1 about the model. The Bayesian network represented by the bipartite graph is converted to a factor graph as follows (example shown in Figure 4.1): First, to allow querying posteriors for indistinguishable protein groups, an additive factor is introduced for all sets of proteins that share the same set of peptides. Then, to reduce loops, save computations, and avoid oscillations later on, we also create peptide cluster factors that calculate and hold the current beliefs for the number of parent proteins for sets of peptides that share the same parent proteins or parent protein groups. Both of these factor types use convolution trees for an efficient adding of discrete random variables[141]. It is worth noting that this hierarchy could also include more intermediate factors but since no sub-quadratic algorithm is known to us to create an optimal hierarchy for the parent protein (sub-)sets the algorithm only uses two levels of aggregation. Also, this aggregation is only performed if the number of contributing proteins or protein groups is greater than one (e.g., peptide E_6 in the

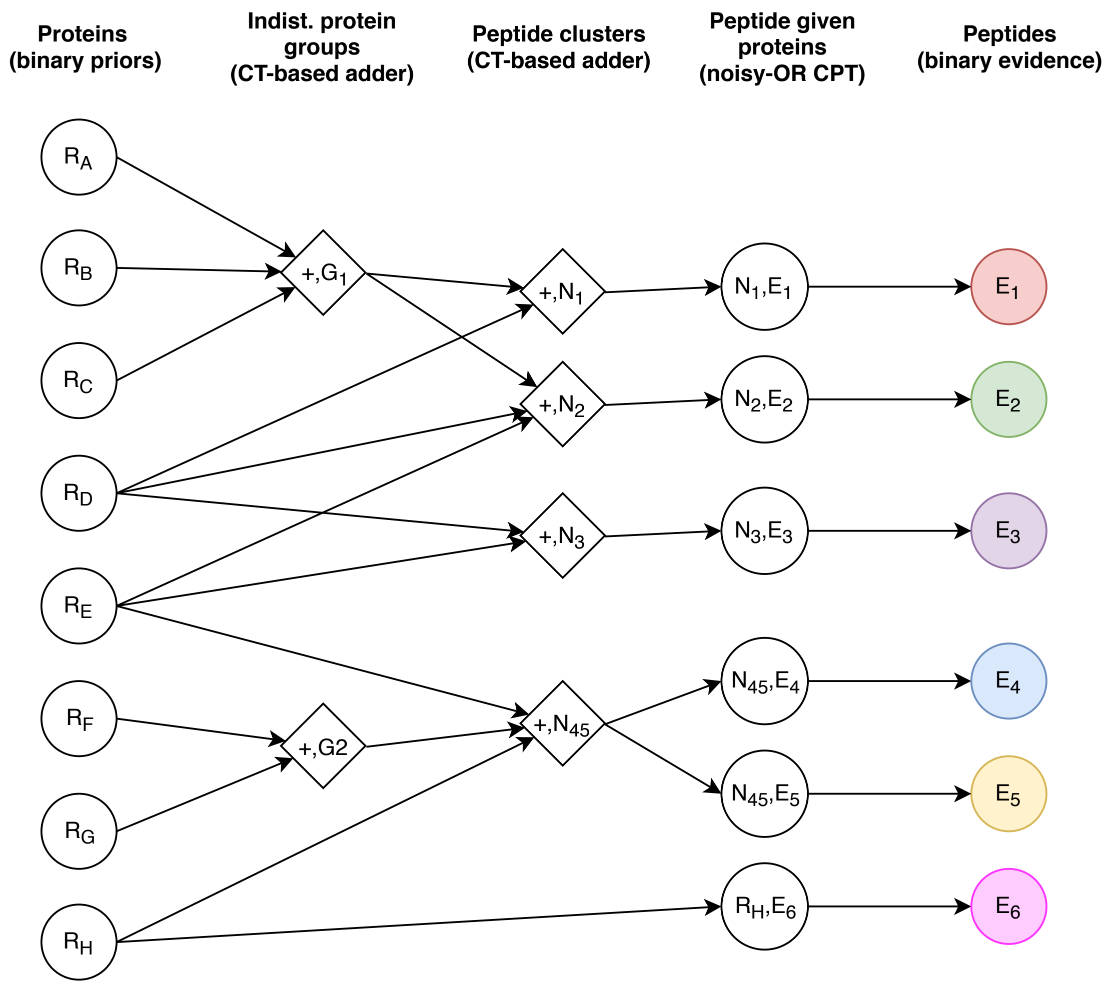


Figure 4.1: The factor graph created for the loopy belief propagation algorithm based on the example in Figure 2.7 (same color and letters). Each node represents one factor. Annotations on the factors describe the set of random variables (RVs) comprising the dimensions of its initial potential and in later stages its updated beliefs. Circles are table-based factors, diamonds convolution-tree (CT) based probabilistic adders. The set of RVs on CT-based adders are implicitly defined by the union of variables from neighbors on the left side of the graph (indicated by a plus sign) and an output variable. Although edges are displayed unidirectional to represent the causality, messages will be passed in both directions.

example). Finally, for each peptide a factor is added to the graph, that holds the CPT for the probability of a peptide being present given the number of parent proteins. The factor nodes on the very left and very right are just singleton factors to keep track of the beliefs on proteins and peptides. They are initialized with priors (parameter γ for proteins) and evidence probabilities (for peptides, e.g., from Percolator) and after convergence of the algorithm hold the final posteriors to be queried. This allows the simultaneous reporting of protein, protein group and updated peptide level posteriors as outcome of the inference on this unified model.

An additional outer loop evaluates the model for (by default 42) points on a three-dimensional grid over the three parameters α, β and γ based on a convex combination of partial AUC (for target-decoy classification) and posterior probability calibration (i.e., comparing posterior based and target-decoy based FDRs). Details on parameter optimization can be found in Appendix B.1.4.

As an optional post-processing, a greedy protein group resolution can be performed. It implements a probabilistic maximum parsimony model, where protein groups are ordered by their posterior probability and starting from the best group, each greedily claims all peptides that it potentially generates until all peptides have been claimed. Proteins or protein groups without any remaining evidence are then deleted or if preferred, implicitly assigned a probability of 0.

4.2.4 Data and data pre-processing

To benchmark the main advancements of EPIFANY and to show the different strengths of the tool we focused on three distinct datasets.

Accuracy and calibration can best be measured on a dataset like the iPRG2016 benchmark data with a set of known ground truth Protein Epitope Signature Tags (PrESTs)[160]. Two sets (labelled A and B) of known PrESTs were designed to share a large number of peptides, spiked-into an *Escherichia coli* lysate background in three different experiments, then measured in triplicates: one experiment each exclusively containing one of the spike-in sets (A, B) and a third containing both sets of PrESTs (A+B). For all sets, we evaluated the performance on identifying the PrESTs of A/B/A+B while having the full database of spike-ins (from A+B), entrapment proteins (i.e., intentionally absent PrESTs) and background proteins as potential candidates. All experiments contained an equimolar concentration of each PrEST. To avoid confusion, we will simply use the term “protein” instead of PrEST in the rest of the manuscript.

A small Universal Proteomics Standard 2 dataset was additionally analyzed with the same pipeline as for the iPRG data to better evaluate pros and cons of methods in detail.

With the goal to measure scalability of the new tool a third, larger scale dataset was analyzed. It is part of an unpublished study and consists of two measurements of human immune cells on a long gradient at two different time points in duplicates.

iPRG2016 data set

Raw files from the corresponding PRIDE[115] project PXD008425 were converted and centroided with msConvert[17] on all levels. The `fasta` database provided together with the study was used to generate a decoy database through homology-aware (i.e., peptide-based) shuffling of amino acids with OpenMS' DecoyDatabase[129,153] tool. Then spectra were searched using Comet (2016.01 rev. 3)[40] allowing a 10 ppm precursor mass tolerance and one missed cleavage for fully tryptic peptides. As fixed modification we required Carbamidomethylation (C). Variable modification was set to Oxidation (M). After merging the results over replicates (by creating the union of proteins and concatenating PSMs) we added target-decoy annotations on protein- and PSM-level. To obtain better discrimination through Percolator 3.02, we extracted additional features specific for the Comet search-engine (see Appendix B.1) before running Percolator with standard settings for the PSM score re-calibration and basic protein inference activated. For all other methods tested we used the PSM-level posterior error probabilities reported by Percolator as input after filtering them at different levels of PSM confidence. Once, slightly by removing PSMs with error probabilities higher or equal to 0.999 (to be consistent with the defaults in Fido and EPIFANY), once at 5% FDR and once at 1% FDR. For comparability with Percolator which only supports group-level reports, Fido and EPIFANY were run with group-level inference as well, thereby querying posteriors on the (indistinguishable) protein group level (i.e., reporting the probability of at least one member being present). ProteinProphet and EPIFANY then report both, group-level and single protein-level probabilities. PIA's recommended default inference procedure "spectrum extractor" also reports groups. Its "report all" option without any parsimonious steps during inference was additionally tested for completeness.

UPS2 standard

Raw data for the UPS2 standard in solution was downloaded from `http://data.marcottelab.org/MSdata/Data_13/`. The highest concentrated sample (30 μl) was chosen to potentially yield the most discoverable proteins. The converted mzML was re-uploaded for reproducibility[117]. A database with the 48 known proteins in various concentrations and six differently shuffled decoys for each protein was used as provided by the lab. Decoys were inflated to obtain a more stable FDR estimate for this small dataset later on. One set of decoys was masked before the Comet search such that full ground truth information is at no point available for any of the methods. Comet settings were the same as for the iPRG data. The different FDR cutoffs (1% and 5%) at which PSMs were filtered were corrected for the factor of additional protein decoys present. During evaluation of protein FDRs we included the knowledge about both decoys and entrapment proteins again by counting false positives with a factor of $\frac{1}{6}$. The correction is done, to correct for the higher prior probabilities of encountering decoys due to their inflation by incorporating multiple decoys per target (i.e., because of invalidating the equal chance assumption).

Large-scale data

After searching the spectra of four runs with MSGF+[72] (unspecified number of missed cleavages, 10 ppm precursor mass tolerance, fully specific Trypsin/P as enzyme, fixed Methylthio (C) modification, variable Oxidation (M)) against the whole human part of the UniProt database (SWISS-PROT + TrEMBL)[8] (release 2017_06, 70,939 proteins plus peptide-level pseudo-reversed decoys appended) and merging the samples the dataset can be summarized by the following numbers:

- 119,921 proteins (matched by at least one PSM)
- 533,218 different peptide sequences
- 807,663 PSMs (including tied matches due to isoleucine/leucine ambiguities or modification locations)
- 734,522 spectra

Again, PSMs were re-scored and probabilities extracted via Percolator 3.02 by training its support vector machine on a subset of 250,000 PSMs for speed and memory efficiency. Since there is currently no way to run protein inference in Percolator separately, the corresponding options were set to be activated as well. The identifications with MSGF scores as well as Percolator scores and features were made available publicly for reproducibility[116]. To evaluate the computationally most demanding task for the Bayesian approaches, Fido was run in single protein level mode (no groups to exploit symmetries). Fido was run with the only available standalone version[i] adapted to

[i]`https://noble.gs.washington.edu/proj/fido/`

Microsoft compilers that is shipped with OpenMS 2.6[ii]. EPIFANY calculates both levels simultaneously by default. Other parameters were left with their defaults in all tools, except for filtering out PSMs under 0.001 probability in ProteinProphet and PIA (to be comparable with the defaults in Fido and EPIFANY). Times and peak memory usage were measured with the Unix utility `time` on a two-socket Intel Xeon X5570 machine (i.e., 16 possible threads) with 64GB of RAM.

Benchmarked tools and tool-specific adaptions

The tools tested represent a diverse set of algorithms. PIA 1.3.10 was chosen as spectrum level parsimony approach considering shared peptides. Percolator 3.02 represents aggregation-based approaches on unique peptides only. ProteinProphet (compiled from TPP 5.1) was included as a commonly used iterative and pseudo-probabilistic heuristic which considers shared peptides as well. Lastly, Fido (in the version shipped with OpenMS) is the representative of Bayesian methods with a very similar model to EPIFANY's but with a different inference procedure. The selection is also based on other recent evaluations of protein inference methods[6,160].

Since PIA accepts OpenMS' idXML format by default, the only change that was done was a renaming of the PSM score type name resulting from Percolator to OpenMS' posterior error probability type so that PIA accepts the scores and interprets them as error probabilities. Inference was performed with and without PIA's own FDR estimation. Shown in the main manuscript are the results directly on Percolators' PEPs as they yield a fairer comparison.

For ProteinProphet OpenMS' IDFileConverter was used to write the error probabilities from Percolator into a pepXML file that is compatible with ProteinProphet to make sure that all tools start from the same set of scores. The protein FDR estimation procedures of the tools were used whenever possible. For ProteinProphet we used the same FDR estimation procedure as for EPIFANY with a concatenated target-decoy database and the equation $\widehat{\mathrm{FDR}} = \frac{(N_{\mathrm{decoy}}+1)}{N_{\mathrm{target}}}$[79]. Groups are counted as decoy if they consist of only decoy proteins. Reported are always q-values unless stated otherwise.

4.2.5 Availability

EPIFANY runs on all major platforms (Windows, Linux, OSX). It is available under an open-source license (BSD three-clause) at `https://openms.org/epifany`. This website also contains demo data, a manual, binary installers for all platforms

[ii]`https://github.com/hendrikweisser/Fido`

and links to the source code repository. EPIFANY relies on the Evergreen inference library released by O. Serang under an MIT license which can be found under `https://bitbucket.org/orserang/evergreenforest`[140]

4.3 Results and discussion

4.3.1 EPIFANY shows improved identification performance

Identification performance of protein inference was benchmarked using the iPRG2016 dataset which was specifically designed to include almost 200 ground truth proteins and a database with additional entrapment sequences. This allowed a comparison of tools at specific known entrapment protein FDRs with different cutoffs at the PSM and protein level. Further, a more detailed investigation of the impact of unfiltered PSMs on a UPS2 dataset with varying concentrations of known proteins was performed.

Unpooled samples A/B with PSMs filtered at 0.001 posterior probability

Results on the loosely filtered, unpooled experiments of the iPRG2016 study show that EPIFANY (with greedy group resolution enabled) yields the highest count of known true positive proteins among the tested methods (Figure 4.2, top). On sample B, EPIFANY identifies 9.14% more true positives at 5% FDR than the second-best method for this measure, PIA. Considering the low number of missing true positives to be identified, this increase is of even greater importance. On both samples, PIA and ProteinProphet do not perform well on just mildly filtered sets of PSMs. Additionally, a special case occurs for sample B, where ProteinProphet's reported proteins are first found at entrapment q-values higher than 5% and therefore an empty bar is shown. The performance of ProteinProphet might be explained by the different pipeline[31] usually used to create its input. Furthermore, ProteinProphet performs a different and more aggressive protein grouping by not only aggregating indistinguishable groups but also by subsuming groups into more general protein ambiguity groups.

However, since this dataset is limited in the number of known proteins that can be found (191 in sample B, 192 in A), looking at the number of identifications at a certain cutoff does not show the full strength of our new method. Not only does it identify more known spike-ins, it also finds them at overall lower entrapment FDRs than most other methods, and thus at a threshold where it is reporting fewer false positives. This is evident in the quickly rising curve of the receiver operating characteristic (ROC). On sample B, EPIFANY is covering the largest partial area under the curve of all tested tools (truncated at nine false positives equaling the maximum number to reach a 5% entrapment FDR; abbreviated pAUC). On the other sample (A) Percolator slightly outperforms EPIFANY in this regard. This pAUC measure is summarized in the lower part of Figure 4.2.

Together with Fido, EPIFANY is also the only tool reporting all correct proteins of sample B after all—though Fido finds all 191 present proteins at an entrapment FDR of 47% (beyond the cutoff in the figures). The other tools tested on loosely filtered sample B (even without any protein FDR cutoff) completely ignore or filter some true positives, most likely due to missing/insufficient evidence, e.g., missing unique peptides in Percolator (which reaches 187 true proteins on B at its maximum FDR of 56%). On sample A, no method is able to identify all true positives at any protein-level cutoff.

When applied to data without stringent PSM FDR filtering, PIA achieves only consistently moderate pAUCs (around 70-75%). Therefore we performed the evaluation for all tools again with the often suggested pre-cutoff of 0.01 PSM FDR[164].

Unpooled samples A/B with PSMs filtered at 1% FDR

Different methods are affected by pre-filtering on the PSM level in various degrees. While Fido struggles to perform well on 1% cutoffs on both unpooled samples, PIA now slightly outperforms EPIFANY (Figure 4.2) on the pAUC measure of sample B. EPIFANY still finds all known true positives at the lowest entrapment FDR (0.035 vs. 0.04 in PIA) and achieves the best agreement between reported FDR and entrapment FDR (Appendix B.9). The online Supplementary Material of our original paper suggests that EPIFANY's parameter optimization actually missed configurations with almost perfect pAUCs (due to balancing calibration and not having information about the labels of entrapment proteins).

On sample A with a strict 1% PSM FDR cutoff however, it is again EPIFANY (almost tied with Percolator) that shows the overall steepest ROC curve. PIA's performance is first recovering at around 2% entrapment FDR and finds together with EPIFANY most (181) of the 192 known true positives at 5% protein entrapment FDR (Figure 4.2). Percolator behaves robust to additional data from low-scoring PSMs but misses out on several true positives on sample B in both strictly and loosely filtered test cases. Although ProteinProphet's number of identified true positives (TPs) benefits significantly from pre-filtering, its pAUC does not.

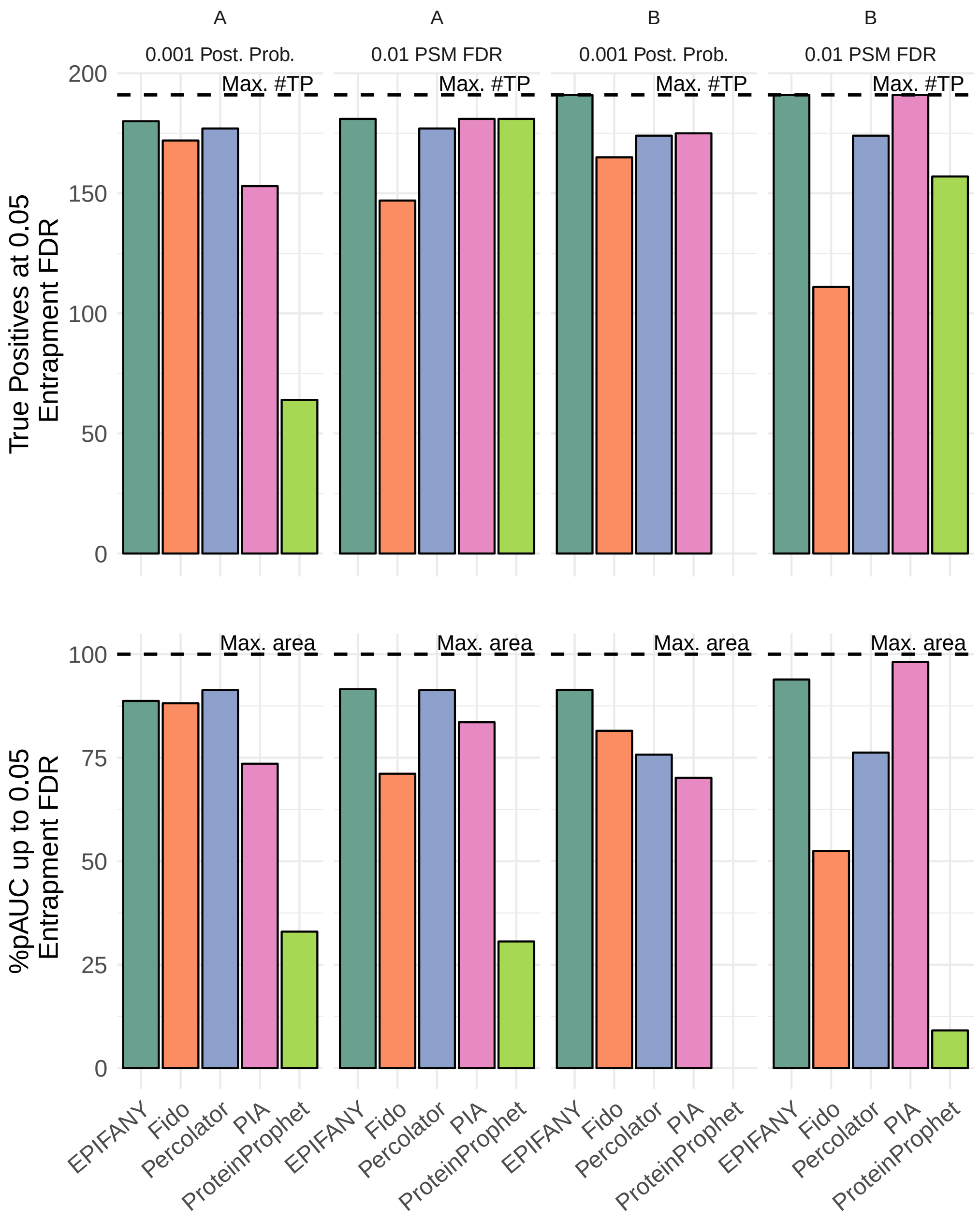


Figure 4.2: Summary statistics on the identification performance of different inference methods at a 5% protein entrapment FDR on the iPRG2016 dataset, samples "A" and "B". PSMs were filtered at two distinct levels (at 0.001 posterior probability and a stricter 0.01 PSM FDR). **Upper:** Number of true positive proteins found. The maximum number of true positives according to the ground truth database given is 191 for "B" and 192 for "A" as indicated by a dashed horizontal line. **Lower:** The percentage of the maximum area under the partial receiver operating curve (%pAUC) as a measure of how quickly methods accumulate true positives at increasing FDR levels until the chosen cutoff at 5% protein entrapment FDR.

Identification performance at 1% protein entrapment FDR

Since a cutoff of 1% FDR on the protein level is often suggested for reporting, we compared performance at this stricter *protein*-level cutoff as well. However, we want to emphasize that due to the small number of known true positives an evaluation on 1% entrapment FDR is now defined based on the ranking of the first *two* false positive entrapment proteins only. This results in both performance measures (pAUC and number of TP) being very similar and overall less robust (Appendix B.20).

The general trend that can be seen at a 5% protein entrapment FDR is also valid at this stricter FDR: PIA (180 TP on sample B; 1% PSM FDR) and Percolator (175 TP on sample A; both PSM-level cutoffs) can reach top identification performance but also can fall behind significantly as can be seen on some of the tested configurations (e.g., max. 126 TP for PIA on A and max. 75 TP for Percolator on B). Fido is robust across samples without strict PSM cutoff (min. 150 TP) but does not perform well at strict PSM-level filtering. Even more pronounced is now the reduced performance of ProteinProphet (max. 55 TP in any combination). EPIFANY shows robust performance across all tested combinations of sample and PSM-level filtering and identifies 147 true positive proteins on sample B and 164 on sample A (independent of any tested PSM filtering), but its full strength here is better reflected in the pAUCs at higher, more stable cutoffs which implicitly include lower ones (Figure 4.2, bottom) or in the full ROC curves shown in Appendix B.4.

Potential advantages of unfiltered PSM input for protein inference

While a filtering on PSM-level before inference indeed seems beneficial to identification performance for at least PIA and ProteinProphet on the subset of known proteins, we argue that in general with Bayesian methods (as with most machine learning approaches), inference on as much data as possible may have several advantages as well.

However, specific examples can be shown more easily on smaller ground truth datasets such as the universal proteomics standard 2 (UPS2) where it can be observed that with full information available, EPIFANY is able to improve the ranking of low concentrated true proteins (e.g., Lysozyme C) compared to some decoys (Figure 4.3). With milder filtering on the PSM level, inference methods can rely on additional data from low-scoring PSMs. Decoys that acquired one or few high-scoring PSMs can then more easily be penalized based on the number of bad PSMs that are simultaneously matching to them. EPIFANY on unfiltered PSMs therefore achieves the highest true positive count at almost all protein FDRs across all analyzed cutoffs and methods

(Figure 4.3). With strict FDR cutoffs on the PSM level, even medium scoring peptides of the true proteins are often not considered since their PSMs do not pass the threshold. Therefore, with stricter cutoffs, a lot of each method's performance depends solely on the ordering among the top scoring peptides of each protein as established by the PSM re-scoring procedures (here Percolator). It becomes harder to misscore the ranking among the best proteins but it simultaneously becomes harder to recover proteins in lower ranks that now resemble decoys even more closely.

Another advantage of less strict cutoffs is that previously indistinguishable proteins (above the desired protein FDR threshold) may become distinguishable. On iPRG2016 sample B for example, EPIFANY on loosely filtered PSMs is able to resolve three additional indistinguishable protein groups (of previously 16 on filtered data).

In any case, good performance on both strictly filtered and almost unfiltered data shows the increased robustness of our method. Figures B.18 and B.19 in the Appendix also suggest that additional PSMs from other sources (in this example wider precursor search windows and additional variable modifications) can be more easily tolerated or even used advantageously in a Bayesian setting (see the curves of Fido and EPIFANY). With those settings, PIA struggles in the very low FDR ranges even with strict cutoffs.

4.3.2 FDR estimates of EPIFANY are more realistic than other estimates

Before introduction of a regularized model and greedy resolution as implemented in EPIFANY, Bayesian methods were shown to report overly optimistic FDRs in the case of datasets like the one tested here[160]. This is due to the fact that on iPRG2016 samples A and B although only a few proteins are known to be present in the sample the spectra were searched against an additional equal number of absent proteins designed to share peptides with the present ones plus an even bigger number of 1,000 absent random entrapment proteins. Unregularized models using standard (sum-product) inference like Fido then would conservatively assign probabilities far from 0 to proteins with similar or no unique evidence which share one or more peptides with truly present proteins. Regularized max-product inference in EPIFANY now makes the assumption that it is less likely for peptides to be generated by many proteins and preferentially distributes the evidence among proteins with the highest unique evidence. Greedy resolution additionally makes a definite, unprobabilistic choice among those proteins based on their posterior probability. By postponing this decision until the end of an identification pipeline, however, reliable uncertainty estimates are available up to the very last step. Even compared to conservative methods ignoring shared peptides for FDR estimation (e.g., Percolator), FDRs reported by EPIFANY are often closer to

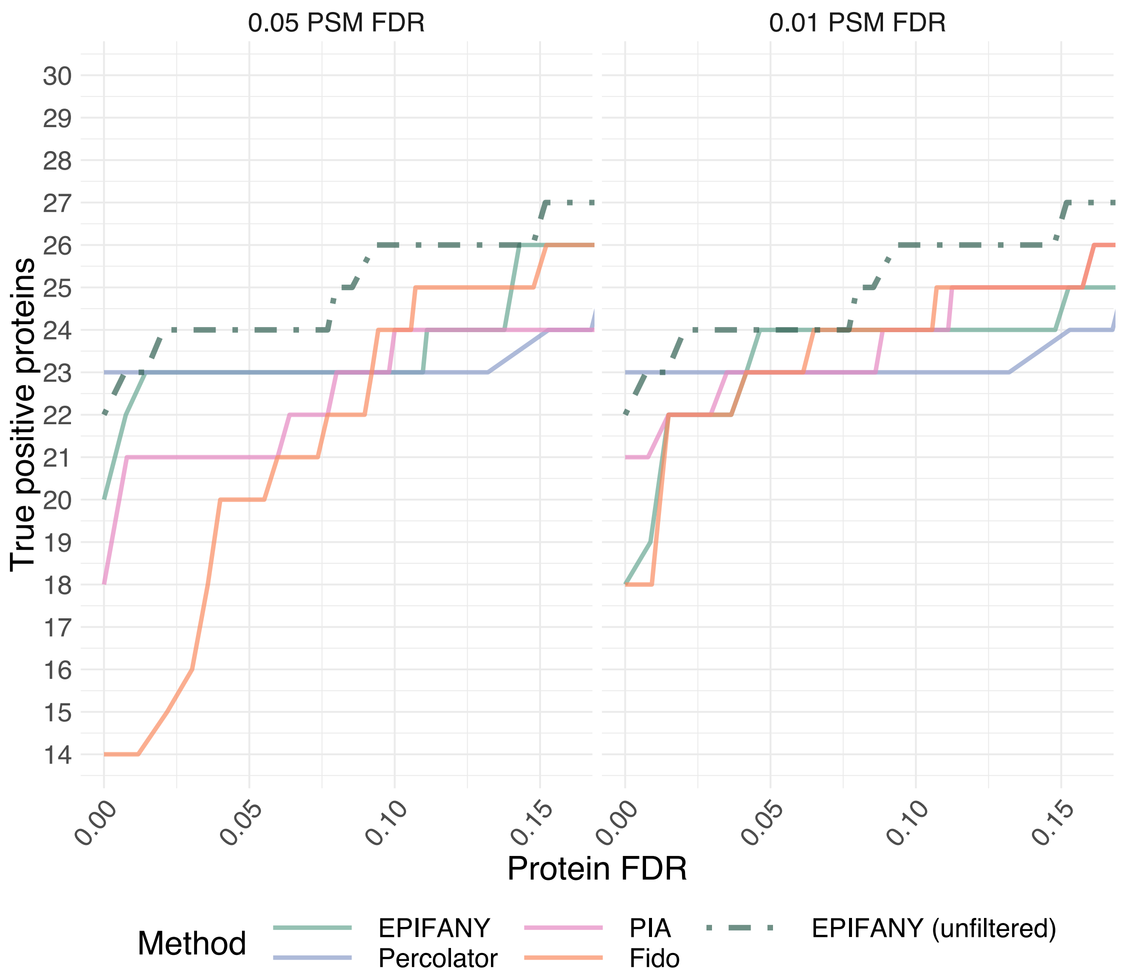


Figure 4.3: Number of true positive UPS2 proteins (max. 48) plotted against the FDR on protein level. No protein groups were present. The subplots from left to right show the results of protein inference on PSMs filtered at increasingly strict cutoffs. **Left:** PSMs were filtered at 0.05 PSM target-decoy FDR. **Right:** PSMs were filtered at 0.01 PSM target-decoy FDR. The results from a run of EPIFANY on completely unfiltered data was overlaid as a dashed line over both of the subplots. Note that the y-axis was cut at the minimal number of true positives observed.

the known entrapment FDRs (also across different PSM cutoffs on A and B). The differences between reported FDR and observed entrapment FDR can be seen e.g., for barely filtered sample B in Figure 4.4 which shows them up to a 15% cutoff (since higher cutoffs are usually not of interest). The cutoff was set this high to generate more robust measures of calibration (by being able to include more point estimates for each method). For sample A, the increase in calibration compared to Percolator shows mainly in higher FDRs but it calibrates as well or better than PIA with simultaneously improved identification performance. Additional pseudo-ROC curves and calibration plots for unpooled samples A and B, different cutoffs and the full FDR ranges can be found in Figures B.14-B.10 in the Appendix.

Additionally, with the optional greedy post-processing step turned off, regularized inference in EPIFANY on its own offers a balance between better calibration (at least in low entrapment FDR ranges up to almost 2%) on the one extreme (A or B) and identification performance on the other (A+B), where it recovers remaining known true proteins between entrapment FDR ranges of 3 to 8% (comparing Figures B.9-B.14 vs.B.15-B.17 in the Appendix). This is due to the fact that proteins, which were identified through shared peptides alone are then assigned a lower probability than the proteins with additional unique evidence but still outscore many of the decoy or false entrapment proteins. Heuristic approaches like greedy, parsimonious and "unique peptide" methods can not model the actually present sharedness in such datasets. "Report all" methods (e.g., through the corresponding option in PIA) perform well on that extreme but are not an alternative for datasets deviating from this very optimistic assumption.

4.3.3 Scalable algorithms allow application to large-scale datasets with vastly disparate discoveries

A common challenge with inference on generative Bayesian models was their scalability due to the speed of the method, which is inherently tied to the complexity of the underlying model. The efficiency of EPIFANY enables full Bayesian inference on problem sizes that were previously intractable given the used model. On the loosely filtered human dataset searched based on the full UniProt database even non-Bayesian approaches struggle with the high connectivity in the resulting protein-peptide graph. This is evident in Figure 4.5 which shows the runtimes and memory consumption of the different tools. Fido took longer than a day of runtime (33.5 hours), ProteinProphet did not finish after a week and also PIA required more than two days of processing (without considering compilation of the intermediate graph format used by PIA). While

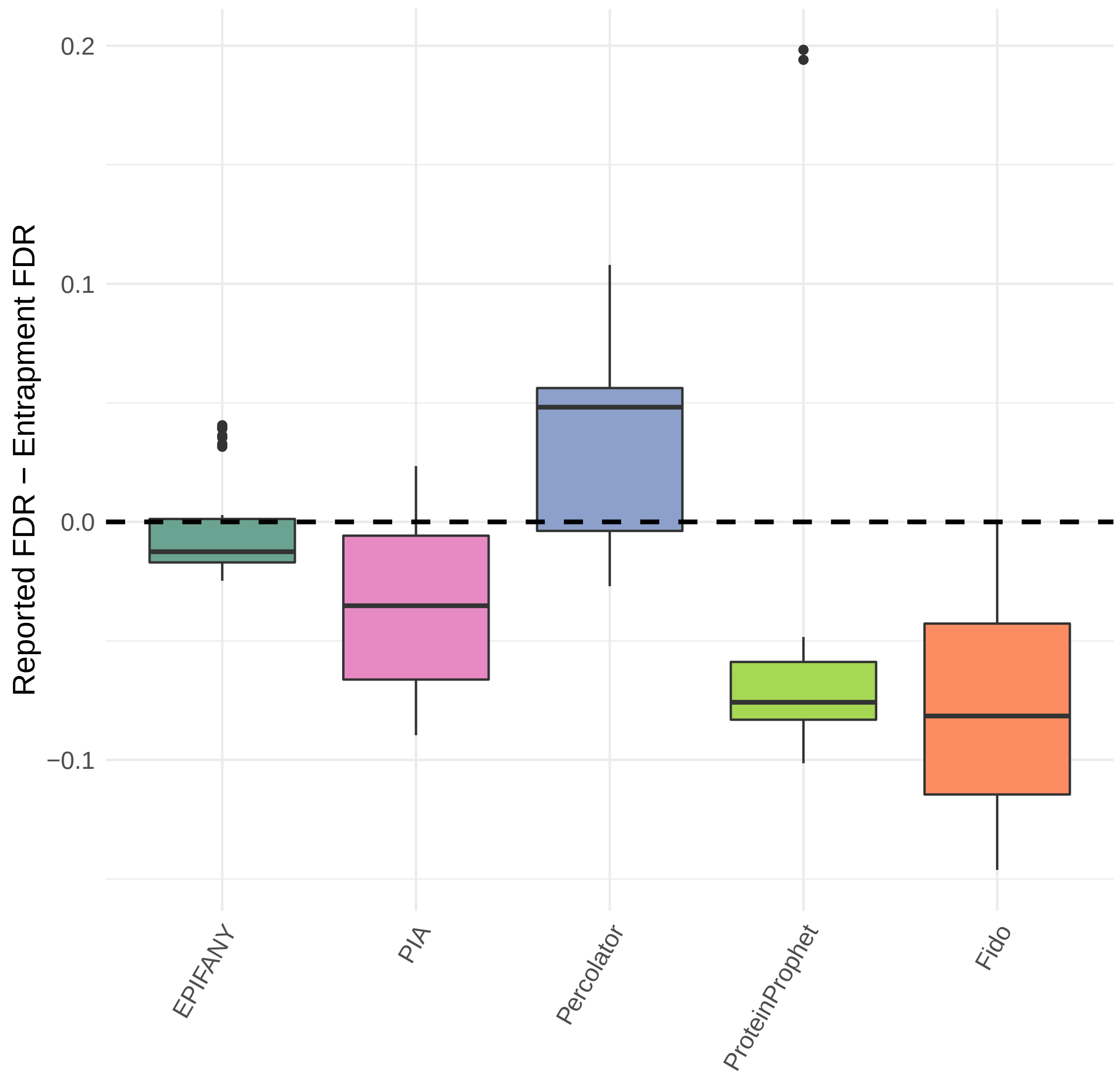


Figure 4.4: Absolute deviations from reported FDR from different tools to the true entrapment FDR (on the subset of proteins with known presence/absence) on the iPRG2016 dataset (experiment "B" with a 0.001 probability cutoff on PSM level). The boxplots aggregate data points for each method until a 15% entrapment FDR. Lower deviations imply better calibration with perfect calibration being indicated by the dashed horizontal line. Data above the horizontal line signify conservative estimates while data below the line signify overly optimistic estimates.

multi-threaded EPIFANY (16 min) is even faster than (subset-trained) Percolator (33 min), it should be noted that most of Percolator's runtime consists of peptide rescoring and internal training of a support vector machine model. The same argument holds for memory usage. Although running Percolator just for protein inference is to our knowledge currently not possible it is likely that the actual runtime on that dataset will be in the single-digit minutes. EPIFANY and Fido were both run with parameter optimization enabled. Runtimes for EPIFANY are roughly linear in the number of parameter sets tested although there are sets that lead to more oscillation and therefore may take longer than others. Concerning multi-threading, none of the methods except for EPIFANY has the option to pass the number of threads to be used. However, it seems from the log that PIA automatically distributes work across all of them. Percolator supports automatic multi-threading across three threads. EPIFANY was once run with one thread and once with all 16 threads enabled. A full 16-fold speed-up is yet far from being achieved due to the different sizes in connected components processed by each thread and the required synchronization during parameter evaluation. For completeness, Fido on the easier problem of "group inference only" took 5.5 hours less (28 hours).

Lower theoretical complexity than exact Bayesian inference methods

To show the theoretical complexity advantage of loopy belief propagation over exact methods like junction tree inference or brute-force enumeration, we investigated the connectivity of the bipartite graph in a very comprehensive dataset of identified peptides in human lung tissue that was downloaded from entry `PRDB000042` of ProteomicsDB and is part of the recently published draft of the human proteome, which is one of the largest data sets analyzed to date[170]. It consists of 32,500 putatively identified peptides (1% PSM-level FDR filtering) from 16,773 possible proteins. Assembling the bipartite graph on the data set resulted in 117,994 edges corresponding to a peptide shared by around 3.63 proteins on average. However, the largest group of proteins sharing the same peptide consists of 124 proteins (for the fairly large peptide `YLENGKETLQR`). It is therefore shown in Figure 4.6 that even if we could avoid brute force enumeration of the complete connected component by building a junction tree that creates more but smaller subproblems, the largest subproblem we would have to solve still requires a number of computations that is in the order of 2^{125} (Figure 4.6). That is the reason why all currently published Bayesian methods need to resort to approximations like pruning, grouping or sampling (at least when encountering a certain complexity limit in a component) and why most models stick to the simple bipartite graph inferring

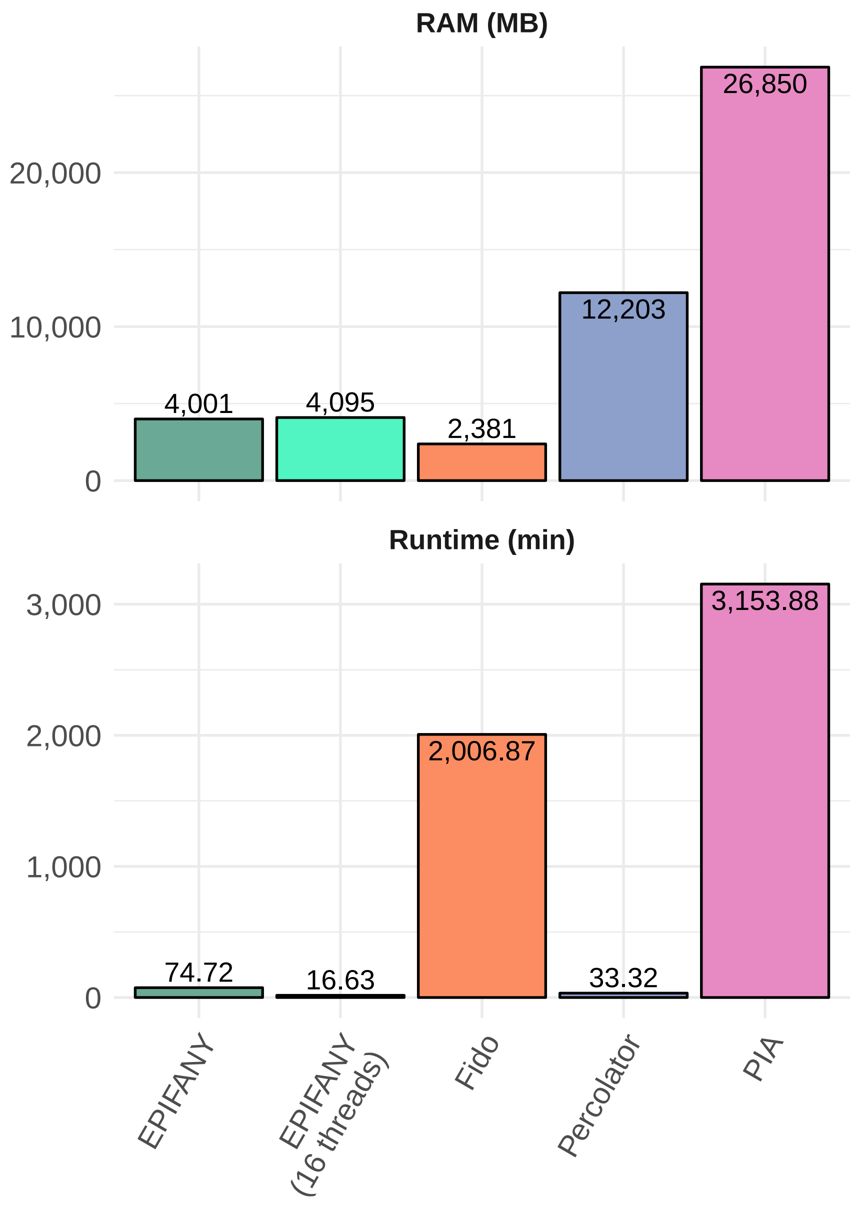


Figure 4.5: Memory usage in megabytes (upper) and runtimes in minutes (lower) on the large-scale human dataset filtered at 0.001 PSM probability. ProteinProphet did not terminate within a week of runtime and was thus not included in the figure. EPIFANY uses one thread by default. Another bar represents EPIFANY with multi-threading on 16 threads enabled via the threads parameter ("EPIFANY (16 threads)".)

absence and presence of proteins with only peptide probabilities as input. Under the reasonable assumption[98] that in a problem of difficulty n, at most n iterations of loopy belief propagation are needed for the results of different clusters (in a connected component) to converge, we show that with our method this leads to a compelling shift to lower complexity problems (green series of bars; Figure 4.6). The hardest component requires approximately 2^{22} operations, as opposed to 2^{125} (with exact junction tree inference without convolution trees; red series) and 2^{379} (with brute force enumeration; black series).

The increasing scale of data sets, as well as the trend toward aggregating seemingly unrelated data sets published by different labs[92] to construct reference proteome maps makes computational efficiency of any Bayesian method absolutely paramount. Loopy belief propagation provides this scalability by passing low-dimensional messages only and performing efficient marginalization via convolution trees.

Scalability in practice: runtime linearly increasing with graph size

Except for the lower theoretical complexity of the algorithm compared to methods like Fido with a similar model, it is hard to make absolute statements about the scalability of its runtime due to its dependency on multiple factors (e.g., the size of connected components, their connectivity and convergence; Appendix Figure B.21).

In general, it starts with a bigger overhead than all the other methods due to the allocation of the tables and messages for every parameter set. Even very small components where posteriors could be quickly calculated naively are currently processed with the general LBP algorithm. This leads to disadvantages on small and/or low complexity datasets (e.g., on filtered iPRG2016 A+B EPIFANY needed 86 seconds instead of 2.5-36 seconds like the other methods on a MacBook Pro 2014; Table B.4 in the Appendix). EPIFANY starts to gain from the theoretical advancements on medium complex data as in the case of loosely filtered PSMs from the A+B sample. Therefore, it already is faster (183 seconds) than PIA (302 seconds) and ProteinProphet (498 seconds). Fido still benefits from the very efficient implementation on small connected components and completes in 80 seconds. For comparison, the full Percolator run on all spectra including inference took 312 seconds (see also Table B.3 in the Appendix). Figure 4.7 additionally shows the scalability of our method based on different posterior probability filter thresholds on the large scale dataset. A linear trend is observed here but cannot be guaranteed in general as explained above.

It is expected that in data sets with more spectra than our human dataset, the complexity and size of separate connected components will at some point reach saturation

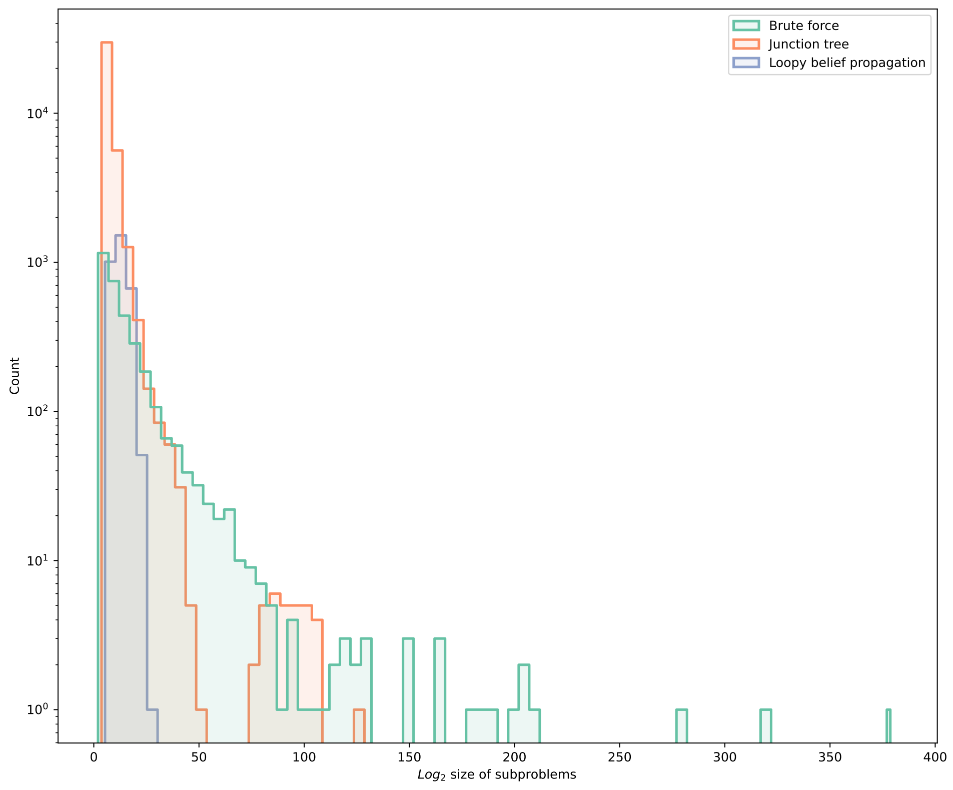


Figure 4.6: Theoretical complexities of subproblems to solve during protein inference with different methods. Plotted are overlapping histograms of the number of subproblems (y-axis; log-scaled) encountered with certain $\log_2$ complexities (x-axis) for different inference algorithms: Brute force enumeration (green), junction tree inference (red) and convolution tree-based loopy belief propagation (blue). Complexities for loopy belief propagation are assuming n messages per edge in a connected component of size n.

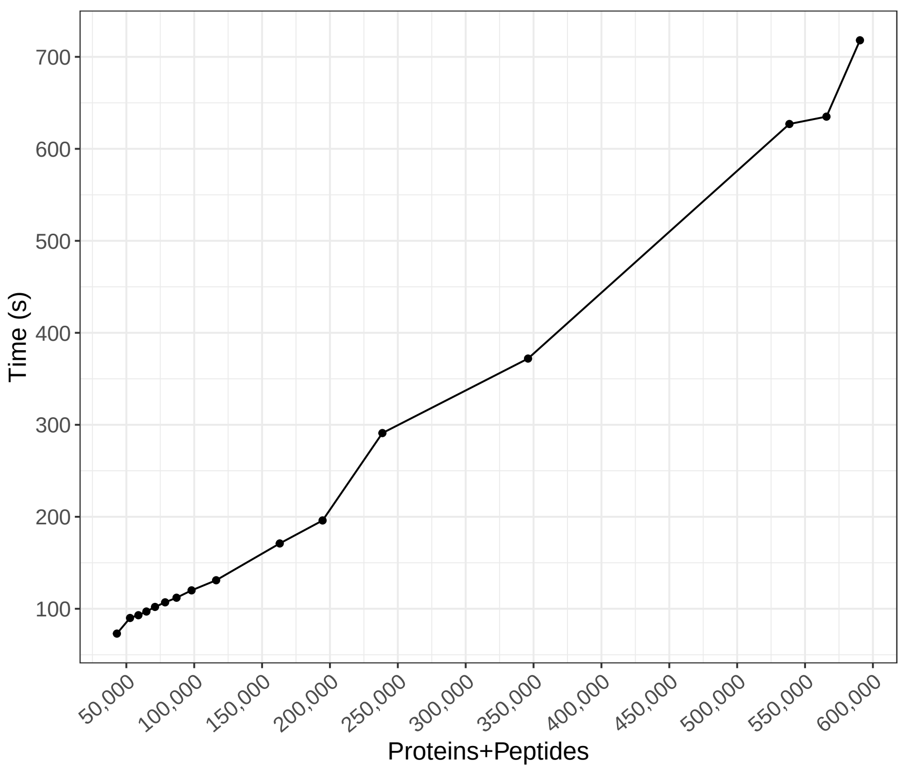


Figure 4.7: Runtime in seconds of EPIFANY (16 threads) for different PSM level thresholds on the human dataset. The total resulting number of potential peptides and proteins was used as the x-axis.

(on the same protein database) such that the growth in runtime of inference methods decreases. More and more PSMs will start hitting the same peptides and form less new paths between previously independent proteins until all peptides are observed.

The efficiency of EPIFANY should allow an application of Bayesian methods also on databases with a higher amount of ambiguities such as proteogenomics or metaproteomics databases as well as in the context of post-translationally modified proteoforms.

We also emphasize, that an application of a Bayesian method incorporating results from shared peptides may lead to different discoveries compared to methods currently used on big datasets that ignore this complication[131,161]. On the largest of the tested datasets for example, Percolator considers only 26,182 of the 51,576 potential target proteins (with at least one PSM above the 0.001 probability cutoff as used in EPIFANY).

4.3.4 Implementation grants flexibility in input and output information

In addition to overall increased performance, EPIFANY's general Bayesian framework also permits the inclusion of auxiliary information (besides PSM data) in a convenient manner. The new implementation allows the usage of arbitrary protein priors for any protein which allows integration of information from complementary RNA-seq experiments, which has been shown to improve protein identification[122]. Due to its modular graph structure it is also easy to add additional probabilistic evidence on the peptide level in the future. Since the factor graph includes both single proteins and indistinguishable protein groups, both results can be output simultaneously. Single protein-level probabilities of proteins in indistinguishable groups are e.g., not possible to be reported in neither PIA—its "report all" option allows that but skips parsimonious inference altogether—nor Percolator. Another novelty is the reporting of updated peptide-level posteriors that can be used to re-score and improve FDR on peptide-level, yielding increased target-decoy classification performance by up to 6% (measured on the full target-decoy based AUC). This effect comes from the fact that the information from sibling peptides[108] in the graph was now propagated and incorporated into every peptide posterior.

4.3.5 Limitations

Since the algorithm's grid search explores multiple parametrizations of the model, there might be settings in which at some point during LBP contradicting messages are generated that have disagreeing beliefs about the probability of a protein or peptide (e.g., from evidence of different parts of the graph). This can often be solved by a step-wise increase in damping in later iterations (see also Appendix B.1.3). In extreme cases, however, it can lead to interruptions in the inference on that part of the graph (since for example zero probabilities cannot be recovered). In the latter case we evaluate the (failing) parameter set by using the prior probabilities of the affected proteins (assuming no knowledge could be gained for those proteins). In the case of non-convergence due to too many iterations the current beliefs for a protein are used. However, those cases are rare and usually seen in very large components caused by extreme parameter sets and evidences only.

Also, due to the fact that the parameter estimation is based on target-decoy annotations, our method is affected by the composition of the decoy database. However, as shown in Figure B.8 in the Appendix different random shuffles of the iPRG2016 database resulted in comparable identification performances with a median partial AUC of 92% and a median absolute deviation of one percent point.

Furthermore, group-level inference is still based on *experimentally* indistinguishable proteins which hinders reproducibility of the specific groupings across multiple runs.[147] If other types of groupings should be performed this has to be reflected in the input before running inference (e.g., by merging protein IDs beforehand).

4.4 Conclusion and Outlook

With EPIFANY we present a new approach to efficient Bayesian protein inference in proteomics that combines excellent inference quality with good runtimes. We also showed that inference on unfiltered data can yield improved identification rates at widely accepted protein FDR thresholds. The underlying method certainly can be improved upon. As mentioned in the Limitations Section 4.3.5, the results depend on the convergence of the algorithm. Convergence is generally affected by the current parametrization of the model as well as the connectivity and evidence in the graph. To which degree still has to be investigated. In case of sub-optimal results on a connected component the algorithm could in the future try different message scheduling types or resort to heuristics. Additionally, the currently experimental options of peptide rescoring and user-defined priors are worth further research. Incorporation of additional evidence especially from MS1-level, replicates or multiple PSMs is a viable extension, too, however, initial tests with precursor mass and retention time deviations yielded noisy, generally disappointing results so far. Regularization of protein groups could be improved by facilitating user-defined protein groups (e.g., by gene or theoretical digest). Together with a re-introduction of proteotypicity with discretized α parameters per peptide instead of a single α per dataset it could help in the discrimination of otherwise indistinguishable proteins. In general, parameter estimation via grid-search is a very time-consuming part of the algorithm and although different parameter sets can be distributed across machines on even larger-scale data, learning speed might be improved by learning on a subset of the graph or completely circumvented by including the parameters as hyperparameters into the probabilistic model. Concerning the impact on quantification, it should be noted that optionally, greedily assigned shared peptides could now be used in quantifying tools as well. While this procedure boosts the number of peptides to be used for quantification it should be handled with care since those quantities might be distorted due to actual contributions from the greedily removed proteins. It is then even more important to use quantification approaches that filter peptides with heavily disagreeing profiles[175]. Another promising approach would be to incorporate peptide quantities into the model e.g., in a way that they influence the degree of regularization for each peptide. After correction for their response factors, shared peptides that show significantly higher intensities than the unique peptides of a protein are more likely to have been produced by multiple parent proteins. This might also help to estimate when a complete greedy approach on a dataset is too conservative.

Chapter 5

Automated workflows for large-scale protein inference and quantification

5.1 Introduction

To make the most out of advanced algorithms for protein identification and quantification, it is important to choose compatible tools (including their parameters), chain them in the correct order and to apply statistically sound error control on various levels. Although the OpenMS Pipeline (TOPP)—in which the developments of this and the previous chapters were integrated—provides all necessary means to do that, the toolset was designed with a strong focus on flexibility. This means it interfaces with many external tools and each TOPP tool has a specific narrowly scoped function. With a given number of (equally compatible) alternatives at each step in a pipeline, the number of possible pipelines to build rises exponentially with pipeline length. Additionally, possible variations of parameters in a tool often require changes in the parameters or even affect the compatibility of downstream tools. Therefore, it becomes more and more difficult and error-prone especially for novice users to perform a start-to-end analysis of their proteomics data.[93] Although the advent of GUI-based workflow systems (such as the Konstanz Information Miner (KNIME) and Galaxy[1]) lowered the barrier for new users to construct analysis pipelines and improved the time needed for construction in general, their support in choosing compatible tools and parameters is still limited[111]. This can lead to a variety of similar, sparsely documented pipelines whose results are hard to compare and can easily be misconfigured or misused when requirements are not met.

5.1.1 The one-tool-solution

An extreme approach to solve these problems are so-called one-tool-solutions[26,120] that try to cover the functionality of one or often multiple pipelines in a single monolithic application. While this allows for coherent parameters, inherently compatible algorithms and efficient passing of data, there is a limit of complexity where a single tool with several tabs of parameters is beneficial to the ease of use compared with a well-designed compact workflow. For example, the graph representation of a workflow adds an advantage in documentation and transparency since it provides a first overview on the steps involved. Further, unless specifically implemented, the work performed in single tools cannot be distributed on compute nodes of for example a high-performance computing infrastructure and therefore often lacks scalability for the ever-growing size of datasets.

5.1.2 Compact standardized workflows with meta-tools

This chapter presents an alternative approach that is now pursued in OpenMS and was developed as part of this thesis. The goal was to find a balance between scalability and flexibility versus usability and robustness. This was achieved by introducing another layer of abstraction among the provided set of tools. This layer comprises a subset of "meta-tools" which are agglomerations of single tools to perform core parts of larger workflows. Those self-contained mini-workflow tools can increase the robustness and usability of the replaced part by allowing frequent internal data consistency checks, auto-resolving dependent parameters and hiding the complicated data flow internally as code. While this hinders simple distribution of computations to remote machines for parallelizable tasks, data can now be shared in memory and file input/output (IO) from/to disk is reduced to a minimum. With those trade-offs in mind, such meta-tools can be created for specific parts of a workflow, where robustness is an issue and the speed-up from parallel remote executions is marginal or outweighed by the increased stability. Together with the original fine-grained tools this allows for adjusting the robustness and flexibility of workflows in more detail to tailor them to specific applications or user bases.

5.1.3 Automation and scalability

To allow for automation and usage on headless hardware, two additional requirements have to be met. It is important that besides a GUI, an easy-to-use command-line interface (CLI) preferably with support for pre-defined profiles or configurations is available. Additionally, the software executing the workflow needs to be able to connect to the various different schedulers that are orchestrating the distribution of jobs on HPC clusters or the cloud. One such flexible workflow executor is provided with the nextflow framework[32] for pipelines written in their domain-specific language (DSL) for workflows. Although Galaxy and KNIME servers could also be set up to distribute tasks across multiple compute workers, they are (a) difficult to integrate with existing scheduling solutions and (b) require more complex installation and configuration routines as well as regular maintenance which might be unreasonable for single-user setups or small labs.

5.1.4 A curated and continuously benchmarked workflow for the community

As so often in the broad field of bioinformatics, there are numerous combinations of parameters and tools (plus their order) to analyze the same dataset for the same research question. Benchmarking those workflows usually involves evaluating them on not one but many performance criteria. While some combinations may achieve subpar results (e.g., due to misconfiguration or misuse), others will form a pareto front of workflows that perform similarly well overall but outperform in specific performance measures or on certain datasets. Picking a workflow among them becomes subjective to the weights that one implicitly assigns to the different criteria. While accuracy of the results for the limited number of ground truth datasets and hardware requirements are comparably easy to measure[95,165], robustness across many datasets, ease of use, and long-term maintenance quality[3] are much harder to evaluate due to difficulties establishing objective metrics and the effort of measuring them (potentially over longer periods of time). Nonetheless, these are important factors for a user's decision on a workflow and an essential part of sustainable bioinformatics software.[86] The ten rules for usable software in computational biology as defined by List, Ebert, and Albrecht[86] therefore were valued highly during the development of the workflow that is described in this chapter.

With the rapidly evolving landscape of proteomics tools and protocols, it is crucial that a workflow is adaptable. By opening up the development of a pipeline to the community, this adaptability can be increased in the sense that missing features can be contributed by anyone and—after review and automated testing—included in the same pipeline or a fork of it. A high number of community contributors further improves long-term maintenance and support. However, such efforts need to be coordinated by enforcing strict contributing and versioning guidelines (of both, tools and workflow), continuous testing and benchmarking as well as curation by a core team. A project that enables this for nextflow pipelines and enforces versioning as well as continuous testing is nf-core[42]. Less constrained and mostly unsupervised alternatives for other workflow systems are centralized repositories for community workflows like KNIME Hub[i] and myExperiment[48].

[i]`https://hub.knime.com`

5.2 Methods

To create a robust, but flexible pipeline that includes the algorithms presented in earlier chapters and scales well for large datasets, several changes and additions had to be made to the OpenMS framework first. This section describes those developments and then gives a detailed view on the pipeline. Lastly, it presents how a continuous quality of code and results are maintained in the future across changes in the pipeline, OpenMS, or its external dependencies.

5.2.1 ProteomicsLFQ meta-tool

One of the issues of current TOPP-based proteomics workflows is their quickly rising complexity with increasingly complex experimental designs. To encapsulate the complex logic of which files to align and link based on their quantifiable features and to reduce the number of tools used in a workflow, the ProteomicsLFQ meta-tool was implemented.

Structural and algorithmic extensions to OpenMS

The implementation of the meta-tool included refactoring the algorithms of the involved tools from the source files of their executables into fully fledged algorithm classes to be used from the OpenMS library. This way, they can now be used in multiple different tools and their function wrapped for usage in other languages (e.g., Python via pyOpenMS or C# via loading of the DLL).

Another large change to the library was the introduction of support for protein groups throughout all currently used algorithms and data structures. This allows for the quantification on group-level as well as the calculation of (picked) protein group FDRs and filtering thereby.

For an efficient handling of larger datasets especially during inference with EPIFANY (see Chapter 4), a lightweight graph-based wrapper around the internal identification data structures was implemented that besides the inference task it was designed for, also allows for quick grouping and traversal through the different layers. The wrapper is based on the Boost graph library and represents a k-partite graph, whose layers comprise proteins, indistinguishable protein groups, peptide clusters, (unmodified) peptide sequences, peptidoforms and finally PSMs. A division into peptidoforms is optional. If used, the following layers are added to the graph: peptidoforms of different replicates, their different charge states and lastly divided by their set of modifications.

To benchmark different inference methods and to provide an even faster alternative to EPIFANY, a simplistic aggregation-based inference method was added that combines the PSM scores (e.g., PEPs) across peptides by carrying over the best score and finally aggregates across proteins with one of the following methods: multiplication, summation or best (default).

Optimizations in speed and memory have been made throughout the algorithms used in the pipeline to facilitate large-scale datasets. Those include batch processing during feature finding, multithreaded scoring of extracted isotope traces and on-the-fly conversion and pre-processing to improve interfacing with third-party search engines.

Similarly, the robustness of other algorithms had to be increased to handle the large variety of data encountered during testing. First, additional data sanitization steps were introduced (for example by removing outliers when fitting distributions for posterior probabilities) but also the usage of exceptions was decreased by better handling unknown modifications or enzymes.

A second similarity-based, tree-guided feature alignment order was implemented and feature linking was extended to ignore unannotated feature outliers based on sliding window statistics of high-confidence identified feature matches.

In general, special care was taken to preserve origins and identifiers of files and spectra to allow unambiguous lookup during and after the execution of the pipeline. The same holds for parameter settings used in the different stages, especially the search engines. Tracking them allows downstream tools to auto-determine their parameters and write more detailed output in the end.

Experimental design support

Before the introduction of support for experimental designs in core tools (e.g., for map alignment or quantification), different designs required different workflows, to assure that fraction n was aligned to fraction n across samples and the union of all fractions from one run was used for quantification against another run. To be able to include the required logic in a tool instead of the workflow, a format for the specification of the meaning of spectrum files—and their labelled channels if present—was designed. It requires only the most basic information needed for correct processing but can be extended with information for downstream tools (like MSstats) to produce a compatible output. The design is also similar enough to the more extensive PRoteomics IDEntifications database (PRIDE) sample-to-data relation format (SDRF)[114] such that the relevant information can easily be extracted with our provided official

SDRF converter[i]. This is the basis for a complete re-analysis of an annotated dataset from PRIDE, resulting in a ready-to-(re-)upload mzTab[49] for a “complete submission” which again contains a basic version of the design.

5.2.2 Nextflow pipeline

To retain robustness also during installation and usage of the pipeline we chose the lightweight workflow executor nextflow that abstracts the complexity of the workflow and its—potentially distributed—execution by providing a simple command line interface. Nextflow’s support for a config file, both for workflow- and execution backend-specific parameters allows a one-line execution of the complete pipeline. An additional benefit is the ability to choose between local installations, conda environments or containers of various types to use/deploy the required tools on the target executor(s).

Workflow overview

A graphical summary of the workflow can be seen in Figure 5.1 and is described in detail in the following section.

As input, the workflow needs access to the mass spectra from the experiment, the protein database to be searched as reference and the experimental design in the aforementioned newly developed format. Mass spectra need to be available in either Thermo RAW or the open mzML file format[91]. The protein database is read from a FASTA file. Paths to inputs may contain wildcards and can be local or remote uniform resource identifiers (URIs) (e.g., from a public file transfer protocol (FTP) server) that will be staged to the workspace of the tasks in which they are needed. Input can be specified in two mutually exclusive ways: either by simply giving the paths to the spectrum files and the experimental design separately or by supplying an SDRF file which already contains paths or URIs to the spectra plus information about potential modifications, instrument type, and appropriate tolerances. The versions of the tools described in the remainder of this section are fixed for every release of the pipeline and are bundled in a versioned docker container for reproducibility.

The first step is a check and a potential conversion of all spectra to centroided spectra in indexed mzML format. Another pre-processing step will add— per default peptide-reversed—decoys to the protein database if requested. After that, the spectra are searched against the database with one or more peptide search engines. In the case of an SDRF file as input, many search parameters are read from it and take preference. Parameters for database search undergo consolidation to match the different definitions

[i]`https://github.com/bigbio/sdrf-pipelines`

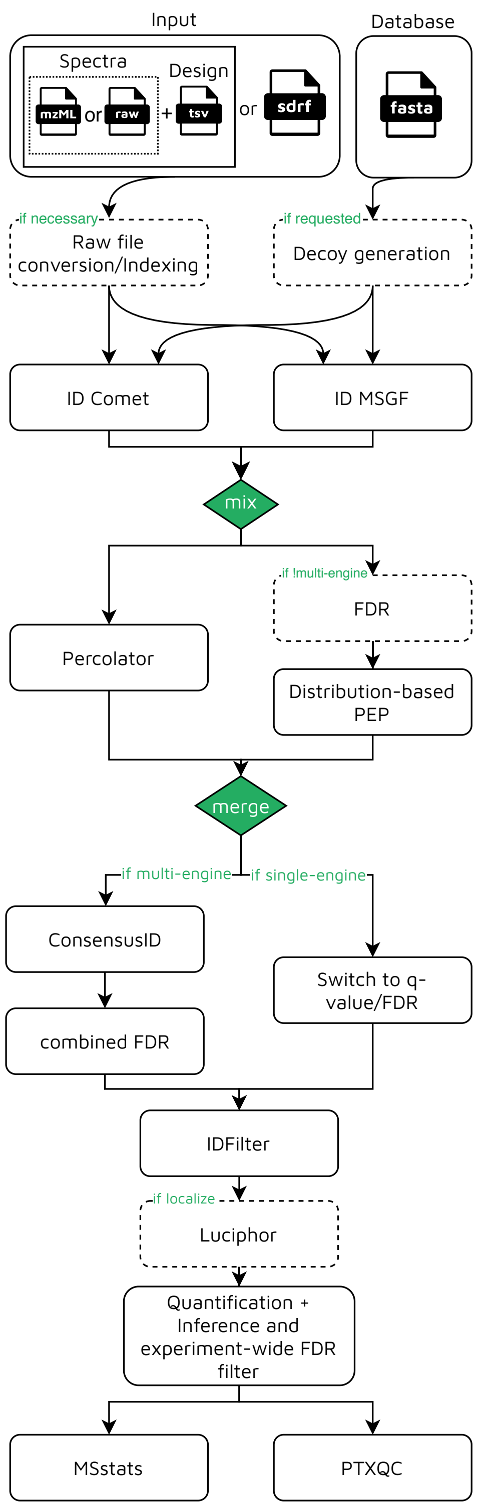


Figure 5.1: Graphical summary of the ProteomicsLFQ pipeline. Dashed boxes are optional steps that are only performed based on the condition displayed in green. “Mix” is synonymous with appendment while “merge” combines files based on an identifier (here the name of the original input file of the identifications).

in the search engines (e.g., different hard-coded enzyme cutting rules, fragment vs. fragment bin tolerances, etc.). To ensure complete and equal mappings from peptides to potential parent proteins across the different search engines, it employs an additional peptide indexing step with OpenMS' internal PeptideIndexer. Subsequently, peptides are re-scored with either Percolator or OpenMS' distribution-fitting approach for multiple purposes:

- It is required to have a probability as input to Bayesian protein inference.
- If multiple search engines were chosen, a probability is additionally needed to bring scores on a comparable scale to combine them with ConsensusID.
- Especially with Percolator, the usage of additional features and a semi-labeled training usually achieves a more discriminative score.

It should be noted that ConsensusID does not introduce a waiting point in the pipeline. As soon as all the search engines and their post-processing are done for a specific input file, ConsensusID and all downstream processes for this file may be scheduled for execution immediately. As a first filtering step, only peptides up to a certain user-defined PSM-level false discovery rate are retained. This is done to speed-up the following steps that depend on the number of peptide identifications (e.g., modification localization, protein inference or targeted feature extraction). It also makes the alignment of related experiments based on identified features less susceptible to noise from wrong identifications. When using Percolator, its own reported q-values are used and also peptide-level cutoffs can be applied. Without it, basic target-decoy FDRs are calculated from the main score of the underlying search engine according to the $\widehat{\mathrm{FDR}} = \frac{(N_{\mathrm{decoy}}+1)}{N_{\mathrm{target}}}$ formula[79] and converted to q-values. With multiple search engines, FDR/q-value calculation and filtering is done based on the consensus probabilities from the combined search. Optionally, the workflow can then continue with LucXor/LuciPHOr2[43] for the calculation of localization probabilities for a given subset of the variable modifications (e.g., for phosphoproteomics studies). Up to this point, the pipeline is able to distribute the tasks for each input file in parallel. Care was taken to optimize the requested resources of each tool in the pipeline based on its ability to benefit from multi-core support and its expected consumption of physical memory.

At the heart of the workflow is the new ProteomicsLFQ tool that uses the post-processed and filtered results of the peptide search engines and the original centroided spectra to infer and quantify proteins under consideration of the given experimental design. The tool starts with some sanity checks on the incoming data (e.g., checks for centroidedness of spectra or the availability of certain scores and decoy information of the IDs) and maps both ID files and mzMLs to the experimental design. It continues with an optional precursor mass correction and calibration and estimates the median full width at half maximum (FWHM) of the most intense retention time profiles for downstream parameters. The next step is a loop over fractions consisting of

- optional untargeted feature finding (with FeatureFinderMultiplex),
- refined, targeted feature finding at confident IDs and the (untargeted) seeds (with FeatureFinderIdentification[168]),
- map alignment to the reference with most IDs (star alignment) or along a guide tree based on confident IDs as anchors,
- grouping of features to be compared, based on an ID- and sample-aware quality threshold clustering approach with a window-based retention time (RT) outlier filtering.

When transfer of IDs is requested, another round of targeted feature finding and chromatogram extraction is applied to potentially fill missing values, where feature linking could not connect any feature to the consensus of the rest of the runs. Seeds for missing features are created at the mean m/z and RT of the apexes of incompletely linked features (which are however found in a minimum percentage of the runs; per default 50%). Since the tool's feature-based quantification only quantifies using basic aggregation of intensities from the most abundant peptides (e.g., top-3 average) and does not provide statistical testing, the pipeline offers a statistical post-processing step with MSstats[20]. MSstats allows for elaborate normalization, a more robust aggregation from peptides to proteins (including outlier removal) and the usage of advanced statistical models to calculate controlled measures of the significance of differences between specified contrasts (of conditions in the experimental design). As an alternative, raw intensities can be returned in a format compatible with the Python-based tool Triqler[159] that uses a single probabilistic graphical model which combines ID probabilities, summarization of quantities and statistics for differential expression.

Reporting

A side product from the standard MSstats script included in the pipeline are small quantification reports with single-contrast volcano plots, heatmaps and per-protein box plots. Another type of report can be created by activating the quality control step of the pipeline. This enables the usage of the package PTXQC[11] to create a detailed and customizable HTML report with important quality control metrics based on the final mzTab from the ProteomicsLFQ tool. This mzTab is also feature-complete in the sense that it contains all relevant sections and metadata, ready to be uploaded to PRIDE as a comprehensive (re-)analysis result for a dataset. Inherently, any pipeline executed with nextflow also has extensive monitoring capabilities resulting in a comprehensive pipeline execution report. The report consists of an execution timeline, a resource usage overview of each step (of RAM, I/O and CPU; see an excerpt in Figure 5.7), a graph representation of the pipeline as well as multiple log files. It can also be monitored in real-time with nf-tower[i].

5.2.3 Testing and community infrastructure developments

The pipeline is hosted publicly on GitHub for transparent, collaborative development and to benefit from other open-source efforts like nf-core[42] and bioconda[50].

Continuous integration and deployment of TOPP tools

An agile development of such a workflow with actively developed tools requires a sophisticated pipeline for continuous testing in the background. For OpenMS and its TOPP tools this was set up on a private Jenkins server as well as with free resources from open continuous integration (CI)/continuous deployment (CD) pipelines like Travis CI and GitHub Actions to balance load and diversify testing environments. The functionality of the library (including their Python bindings) and the built-upon tools (including their wrappers for workflow engines) are tested for every contribution. Installers, pypi packages and containers on DockerHub are additionally deployed every night after successful testing. Bioconda packages (including the resulting biocontainers[29]) are created for every release, to have a stable reference in every release of the pipeline. This consistent and semantic versioning guarantees reproducibility of results for every version. The own Anaconda cloud channel of OpenMS for its latest nightly development versions is registered in the development branch of the ProteomicsLFQ pipeline for seamless integration of the newest features and early discovery of bugs.

Incorporation into nf-core

To make sure, that the pipeline under all its changes—let it be additions or updates of the underlying tools, parameter tweaks, or modifications of the data flow—functions as intended, community reviews and continuous testing is also needed on the pipeline level. Therefore, the pipeline was fully developed under nf-core standards and added to their list of community-curated pipelines. Development then profits from standardized

[i]`https://tower.nf`

templates and executors for continuous testing of small datasets on GitHub actions and large datasets on Amazon Web Services (AWS). Our ProteomicsLFQ pipeline specifically, is tested against a small in-silico fractionated bovine serum albumine (BSA) sample for general functionality and a larger ground truth dilution series of Sigma-Aldrich's standard mixture of 48 proteins with fractions and replicates for its performance and accuracy[123]. New developers and users benefit from the knowledgeable bioinformatics community in nf-core, which also helps in keeping the pipeline up-to-date with newest developments in nextflow and the nf-core-specific tools.

5.3 Results

The main advancements of the workflow described in this chapter include its lean, easy-to-use interface, its applicability and scalability to a wide range of datasets and hardware setups as well as its tight interoperability with PRIDE datasets. All this is provided with state-of-the-art quantification accuracy and performance.

5.3.1 Excellent performance in identifying spike-in proteins based on different concentrations in a public protein standard

To show state-of-the-art performance of the pipeline, it is regularly evaluated on an available ground truth dataset from PRIDE. The dataset PXD001819 is a dilution series of equally concentrated universal protein standard (UPS)1 proteins spiked into a yeast background[123]. The spike-in proteins were injected in nine different concentrations and all samples measured in triplicates to benchmark tools and workflows for label-free quantification such as ProteomicsLFQ.

This section shows the results of this automated evaluation for the current release of the pipeline (1.0.0). The pipeline was run with a cutoff of 0.01 FDR (on both PSM and protein level). To boost the number of quantifiable proteins untargeted feature finding and its implied matching of IDs between runs ("`--targeted_only false`" parameter) was enabled as well. The remaining parameters were solely configured through the input SDRF file hosted on PRIDE or by using default parameter values.

Heatmaps, profile plots and volcano plots from MSstats for all pairwise contrasts are generated by the pipeline per default. Figure 5.2 shows an example of a volcano plot comparing 12500 amol and 5000 amol. The labels were edited manually to cleanly display the proteins for which a ground truth fold change is known. This specific contrast was chosen for display because 5000 amol is the lowest concentration where most of the spike-in proteins can be reliably identified by MS2 spectra and the next higher measured concentration poses the most difficult quantification due to low fold changes.

In this contrast our pipeline reports 46—of 48 possible—quantified UPS proteins (with 36 marked significantly deregulated at p-value cutoff 0.05). The accuracy of the UPS protein fold changes was measured as a root-mean squared deviation (RMSD) of 0.556 around the expected fold change of -1.32, dominated by one outlier as shown in Figure 5.2. To put the raw (relative) quantification accuracy in relation to existing tools, we will compare it to a state-of-the-art software package in the next subsection using all contrasts.

We further evaluated the classification performance of correctly identifying significantly deregulated proteins. For this, we created a receiver operating characteristic (ROC) curve based on the (multiple testing) adjusted p-value as cutoff value (Figure 5.3). The area under the curve (AUC) of 0.948 reveals an excellent performance, regardless of the chosen cutoff (0.01 and 0.05 are indicated on the plot). This means, on the tested contrast our workflow always yields a good trade-off between sensitivity and specificity.

This performance also generalizes to most of the other levels of spike-in concentrations on the benchmark dataset (Figure 5.4). Only at 2500 amol the results worsen due to the increasing difficulty of quantifying peptides around the limit of detection. However, this is a common issue of feature-based quantification and also other established software are struggling at those concentrations as shown in the next subsection.

5.3.2 State-of-the-art raw quantification accuracy at favorable computational costs

To measure the raw quantification accuracy and to compare it to other popular tools for label-free quantification, the same dataset was analysed using MaxQuant[27] 1.6.10.43 (the latest version from bioconda at the time of benchmarking). The same parameters were used whenever possible by utilizing the SDRF-to-MaxQuant converter [i] together with manually set parameters. For comparison with the implicit matching between runs with active untargeted feature finding in the ProteomicsLFQ pipeline, the corresponding option was activated as well in MaxQuant. The full parameter file is available in the benchmarking repository [ii]. A comparison between the results of the ProteomicsLFQ pipeline and the median MaxLFQ intensities across replicates (Figure 5.5) reveals that for spike-in concentrations of 2500 amol or higher, both methods perform similarly in fold change accuracy, except that ProteomicsLFQ is able to compare increasingly

[i]https://github.com/bigbio/sdrf-pipelines
[ii]https://github.com/wombat-p/MaxQuant-Workflow/tree/dev/Results/dev-20210530-JU_DE

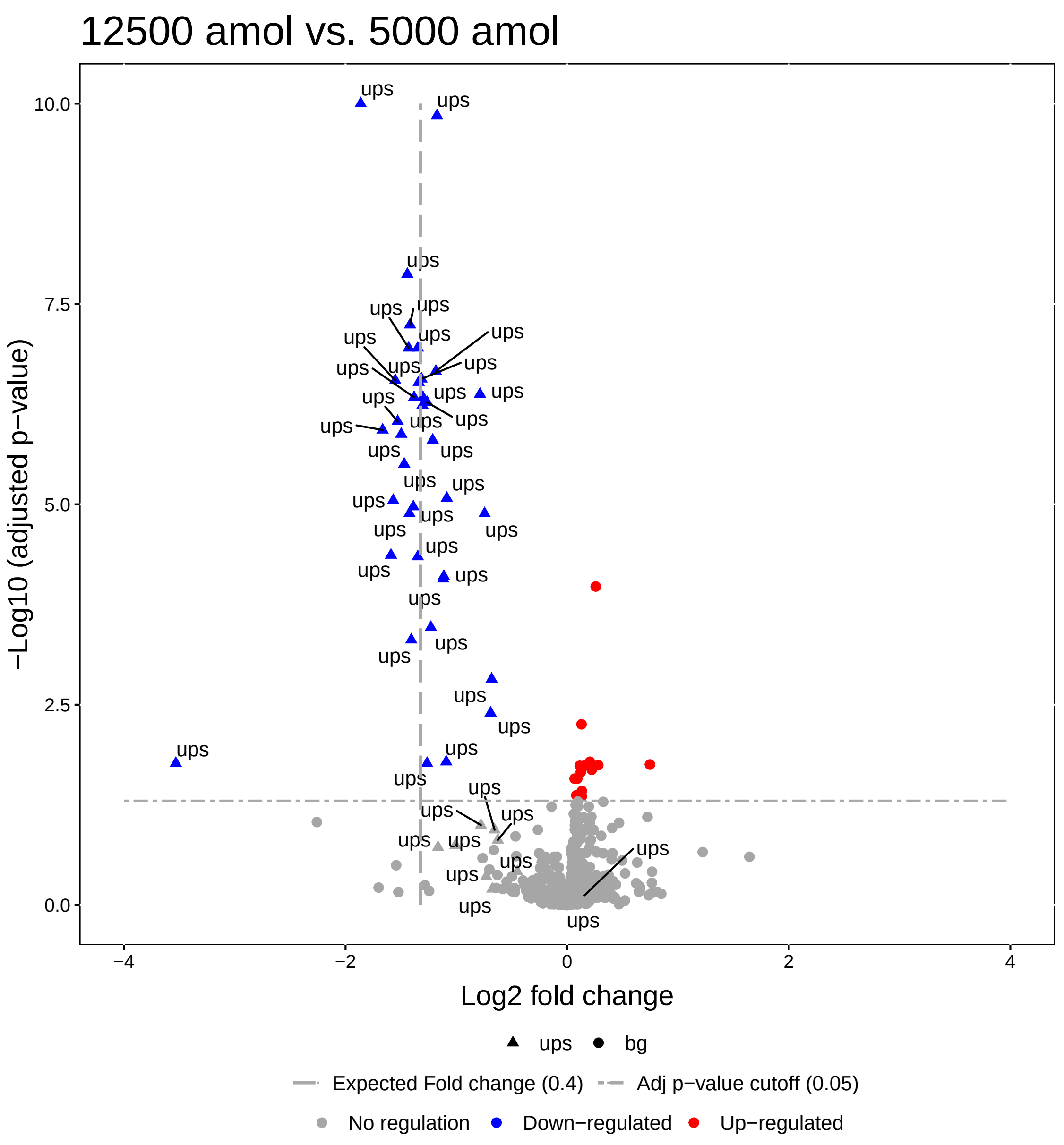


Figure 5.2: Volcano plot from MSstats for the comparison of spike-in concentrations 12500 amol and 5000 amol, adapted to show the expected fold change (vertical dashed line) and the relevant UPS spike-in proteins (labeled “ups” and triangle shaped). At an adjusted p-value cutoff of 0.05 (horizontal dot-dashed line), 36 of the 48 UPS proteins were correctly found to be significantly downregulated. 54 proteins were determined significantly deregulated in total (depicted as colored shapes).

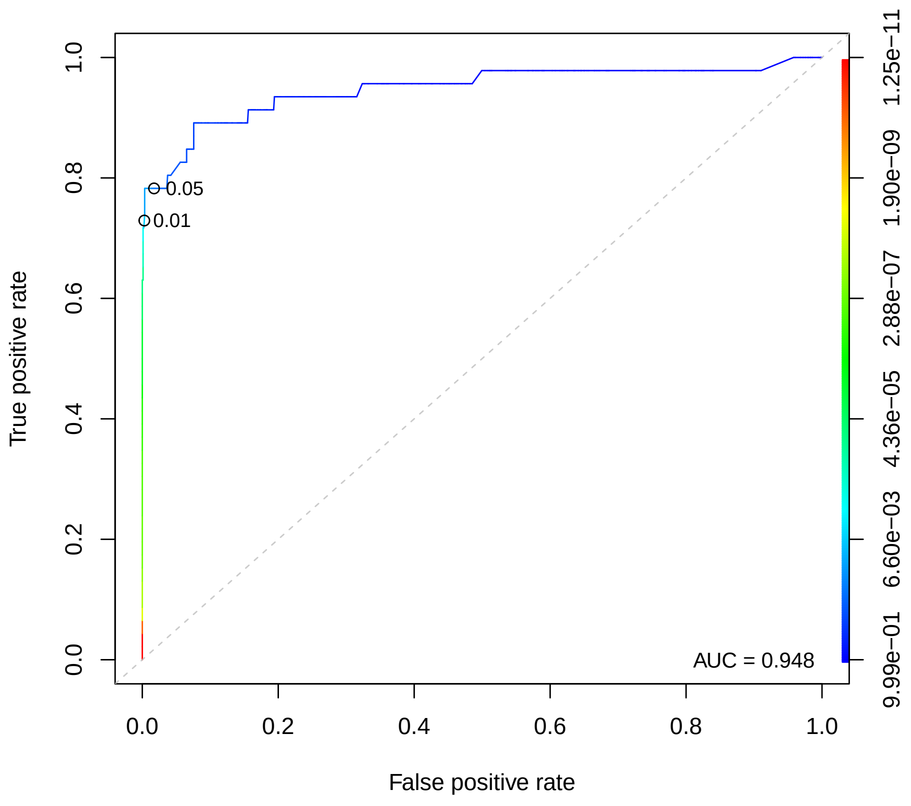


Figure 5.3: ROC curve to evaluate the performance of discovering significantly deregulated proteins in the 12500 vs. 5000 amol contrast of the UPS1 benchmark dataset. The chance line is indicated on the diagonal. The AUC, a combined measure of sensitivity and specificity is noted on the lower right. An AUC of 1.0 means perfect classification, while 0.5 would be achieved by random guessing. Possible cutoff values are color-coded along the curve (see legend) and two common cutoff values (0.01 and 0.05) were specifically marked with points.

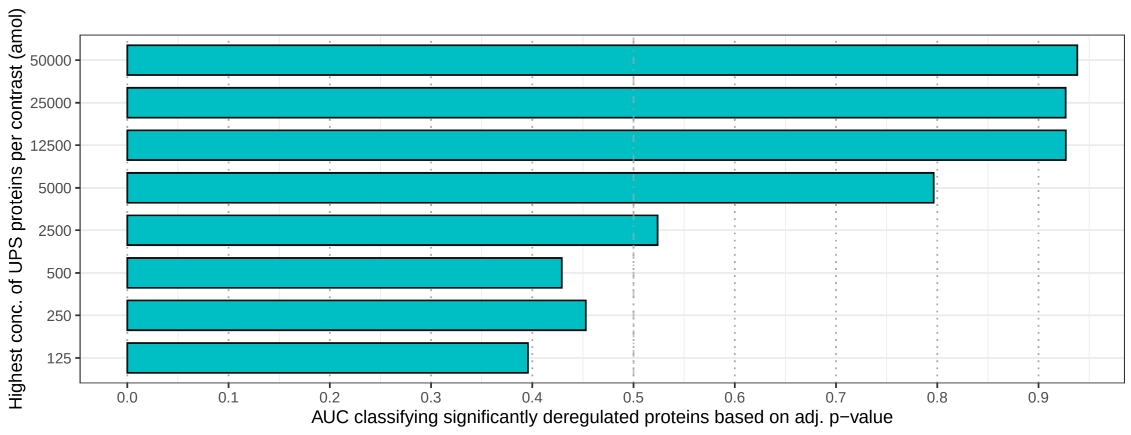


Figure 5.4: Mean AUCs for classifying significantly deregulated proteins based on adjusted p-values as reported by our ProteomicsLFQ workflow. The means were calculated on groups of contrasts that share the same highest concentration (e.g., 25000 contains contrasts 50 vs. 25000, 125 vs. 25000, . . . , 5000 vs. 2500). As ground-truth, all background proteins were considered as truly unregulated, while spike-in UPS proteins should be found as deregulated.

more of the UPS proteins without missing values. MaxQuant loses quantifiable proteins quickly at lower concentrations—note that the smaller the reference concentration, the more comparisons this concentration is part of. However, at 500 amol and below, MaxQuant is more specific in the sense that although it reports finite fold changes for only 4 to 8 proteins per single comparison, it is able to uncover the true fold change for a few of them. In contrast to that, ProteomicsLFQ consistently quantifies more proteins (as with higher concentrations) but underestimates the true fold changes, up to a level where UPS proteins are reported to be unregulated (i.e., without any fold change). This might indicate that during matching between runs, ProteomicsLFQ is too liberal in picking up potential quantitative features in low intensity regions close to noise. This happens either by falsely interpreting the actual presence of a feature or by extending true mass traces too far into noise peaks.

At the same time, ProteomicsLFQ requires less computational resources than MaxQuant, both in terms of compute time and peak RAM usage (Figure 5.6). On our setup—a private cluster on the de.NBI cloud with six worker nodes—ProteomicsLFQ completed in 1h 23m 25s, while MaxQuant needed 3h 31m 12s. The peak memory usage (i.e., how much RAM the largest node in a cluster would need to have) was 5.027 GB for ProteomicsLFQ and 61.390 GB for MaxQuant. Figure 5.6 shows the distribution of the needed resources per step. Note that MaxQuant basically consists of one monolithic step with some minor preprocessing (the generation of the parameter file

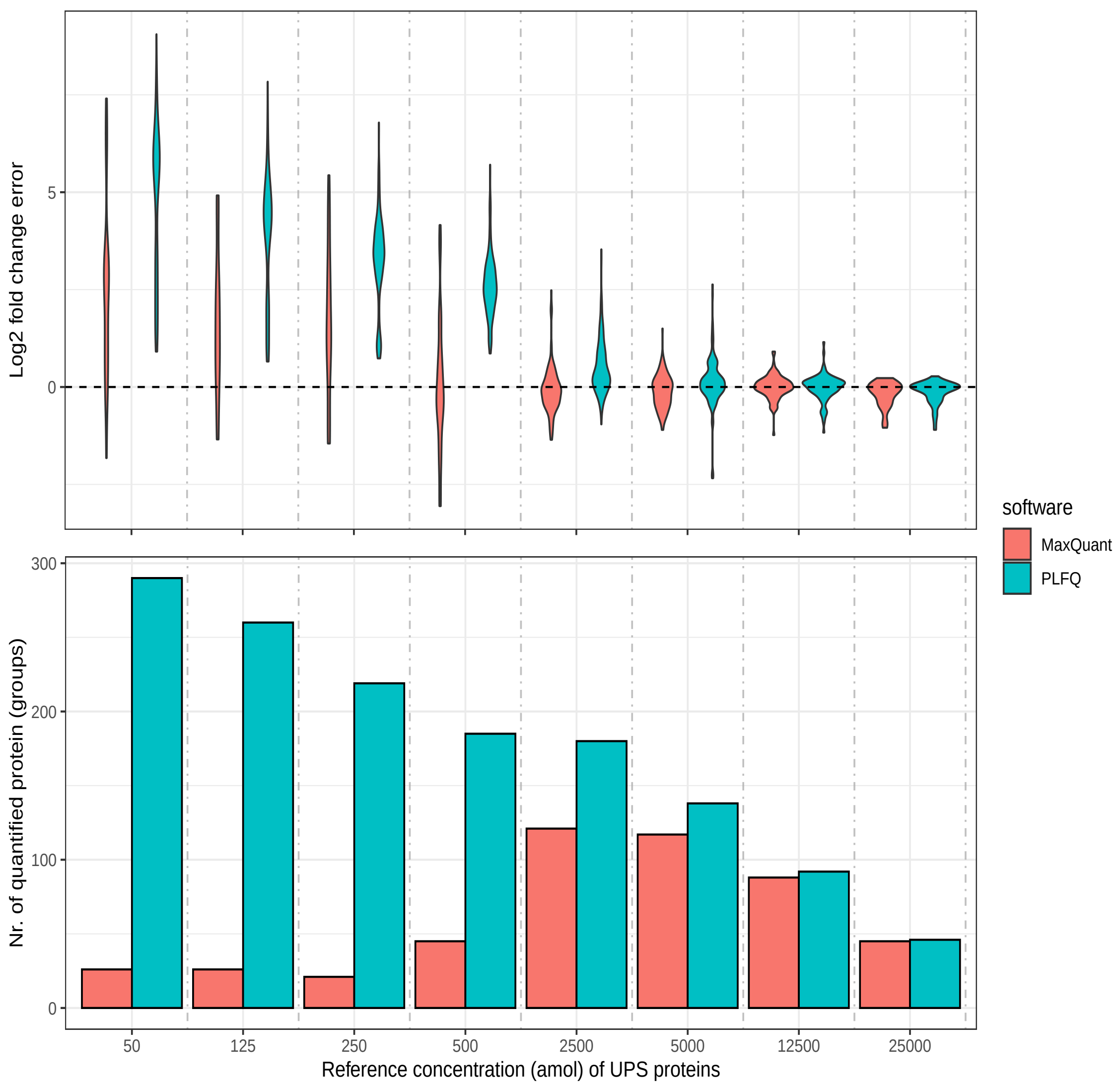


Figure 5.5: Performance indicators over all contrasts of the UPS1 dataset (PXD001819). Reference concentrations are always the lower of the two compared. Fold changes are calculated as $\log_2(\frac{\text{Intensity at reference sample}}{\text{Intensity in higher concentrated sample}})$. Values are summarized over all possible comparisons for a reference concentration for better readability. **Top:** Violin plots of absolute $\log_2$ fold change errors (closer to dashed line at zero means better). The error is calculated as expected minus observed fold change, therefore errors greater than zero mean underestimation. **Bottom:** Bar plot for the sum of the number of proteins or protein groups quantified in both compared conditions. Here, higher is arguably better.

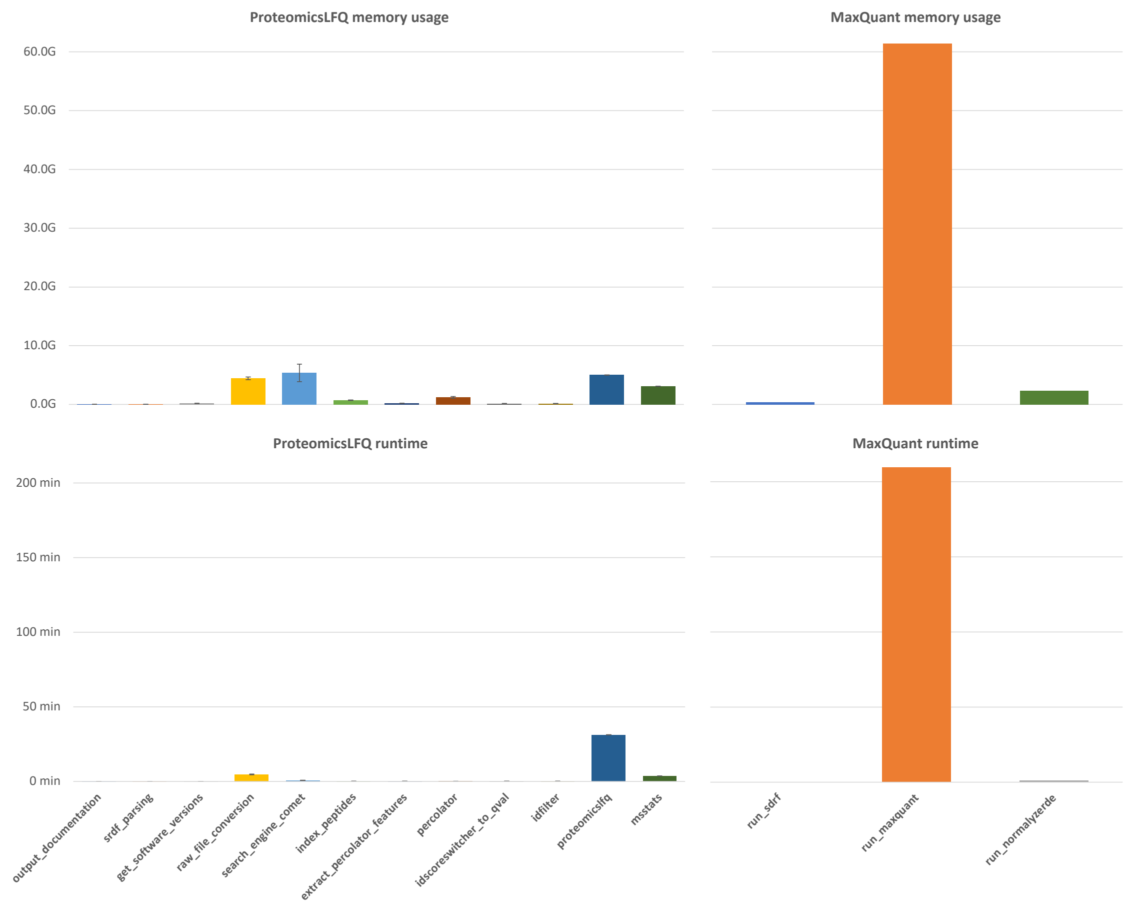


Figure 5.6: Runtimes and memory usage (physical + virtual) of the ProteomicsLFQ workflow and MaxQuant on the 27 samples of the UPS1 dataset. Resource consumption is plotted by process. While MaxQuant only consists of the monolithic "run_maxquant" process plus some minor pre- and post-processing, the ProteomicsLFQ pipeline consists of 12 steps of which some are executed once per input file and therefore their values are represented with error bars for their standard deviation.

from SDRF) and postprocessing (intensity normalization). Contrasting, ProteomicsLFQ could—with enough compute nodes or a computer powerful enough to spin up multiple containers or processes—parallelize the processing of the steps revolving around peptide identification and scoring results even further, up to a maximum of one process per input file. The runtime would then be bounded by the central "proteomicslfq" step, as we will show in the next section on a larger dataset. Even if all processes were run sequentially, the total runtime is around 3h 17m and therefore 15 minutes faster than MaxQuant.

5.3.3 Scalability and robustness shown through large scale re-analysis of PRIDE datasets

To perform a one-command re-analysis of an annotated label-free PRIDE dataset, the only two required parameters are a URI to the SDRF file and a URI to a matching protein database (potentially provided by the original uploader as well). URIs can be local or remote and support protocols like `file://`, `http://`, `ftp://`, `s3://`. For the majority of smaller datasets, the supplied SDRF can be used as-is. datasets with experimental designs including several instruments, different enrichment protocols and/or fractionation setups, for which a direct comparison of feature quantities does not make sense, should be split by the user into smaller tables of comparable runs (e.g., by using the splitting option in the SDRF-to-OpenMS converter[i]. This comfortable usage allowed us to easily re-analyse dozens of datasets on PRIDE with different sizes and designs.

Scalability

The scalability of the workflow was evaluated based on the PRIDE dataset with identifier PXD008722[85]. The dataset consists of 252 raw files from human heart samples of seven individuals measured on the same instrument. Samples come from three different regions of the heart (left/right ventricle and left atrium) and were equally pre-fractionated into twelve fractions. For the size of the dataset in combination with PRIDE's local LSF cluster, the pipeline's default configuration for different process types (e.g., small, medium, long-running) was adapted to accommodate the higher requirements for each submitted job. Other than that, to potentially increase the yield of confidently identified spectra, both currently supported search engines (Comet and MSGF+) were enabled to be combined with ConsensusID. At the same time, untargeted feature finding was activated to allow for transfer of identifications to unidentified features between runs during quantification.

The full pipeline finished after 34h 6min 44s, with the ProteomicsLFQ meta-tool clearly dominating runtime (31 hours) and memory usage (52.5 GB) as a single process (Figure 5.7). This is mainly due to the fact that identification-based feature detection, alignment and matching between runs with subsequent inference needs information from the union of all identifications as well as alignable subsets of the peak maps to run. However, when comparing against the cumulative requirements of e.g., the 252 single MSGF+ search engine processes (median: runtime 15 minutes and 8 GB of RAM), we argue that the consumption of the combining meta-tool is justifiable

[i]`https://github.com/bigbio/sdrf-pipelines`

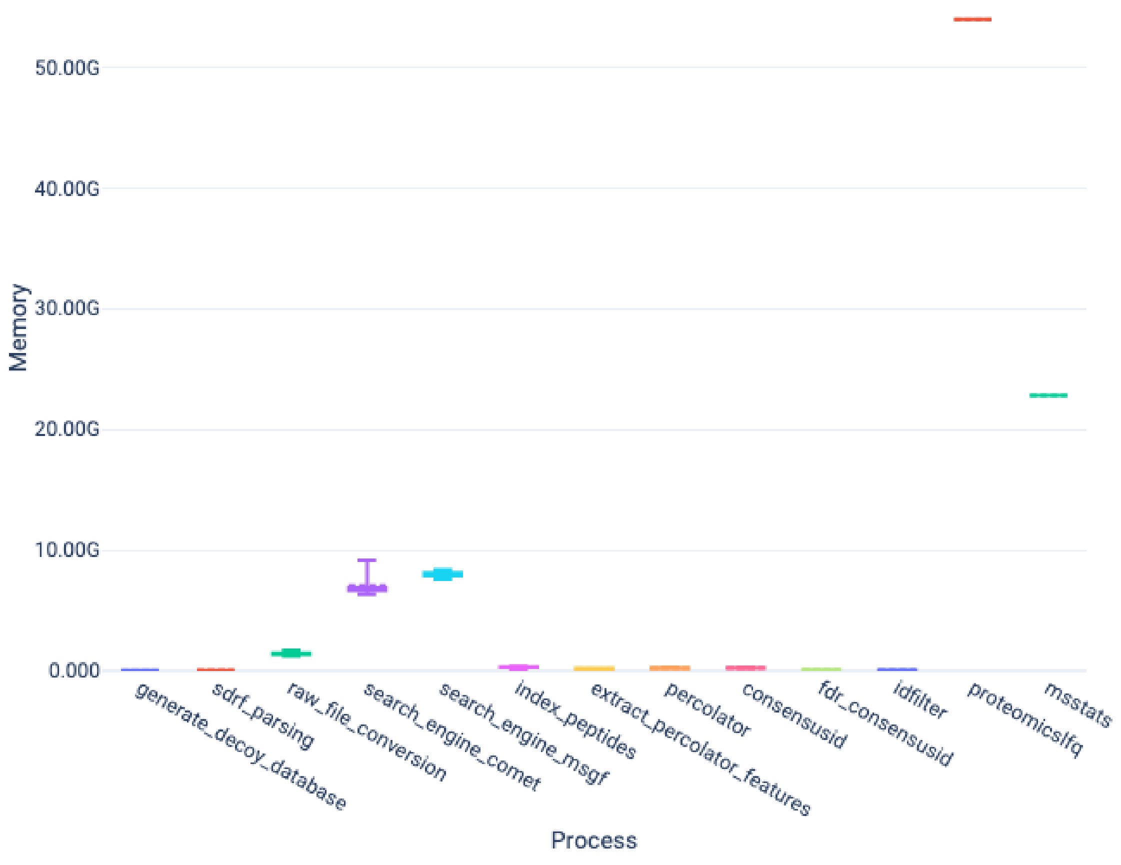


Figure 5.7: RAM usage in gigabytes of the different processes in the proteomicslfq pipeline for dataset PXD008722, consisting of 252 raw files. For steps that are run multiple times, ranges are depicted by whiskers around the plotted boxes.

and difficult to avoid with efficient in-memory processing. In general, peptide search, rescoring, modification localization as well as confidence estimation and filtering can be parallelized easily and scale well with multiple compute nodes or containers. In that part of the pipeline, linearity in scaling with number of available compute nodes or max. number of parallel containers is achieved up to the number of input files, considering file sizes and number of identifications are roughly equal between each of them. The ProteomicsLFQ step is per default assigned to a higher resource class and will request a high-memory node with a higher number of processors to make use of the parallelization of the program on the in-memory datastructures.

Robustness

As a first benchmark of its robustness for large-scale re-analyses of different datasets, we collaborated on the curation of currently over 200 SDRF annotations of public datasets[i] on PRIDE. The selected studies include proteogenomics applications, phosphoproteomics studies and “standard” proteomics experiments with various experimental designs. As of today, as much as 57 datasets have been successfully re-analyzed. A table of successfully re-analyzed identifiers can be found in Appendix C.1.

5.3.4 Simple usage due to user-friendly launching interfaces, auxiliary supervision tools and QC reports

After installation of nextflow and preferably a containerization software, an additional installation of the nf-core tools is recommended for easiest use. After that, the pipeline can be downloaded, configured and launched with a single command:

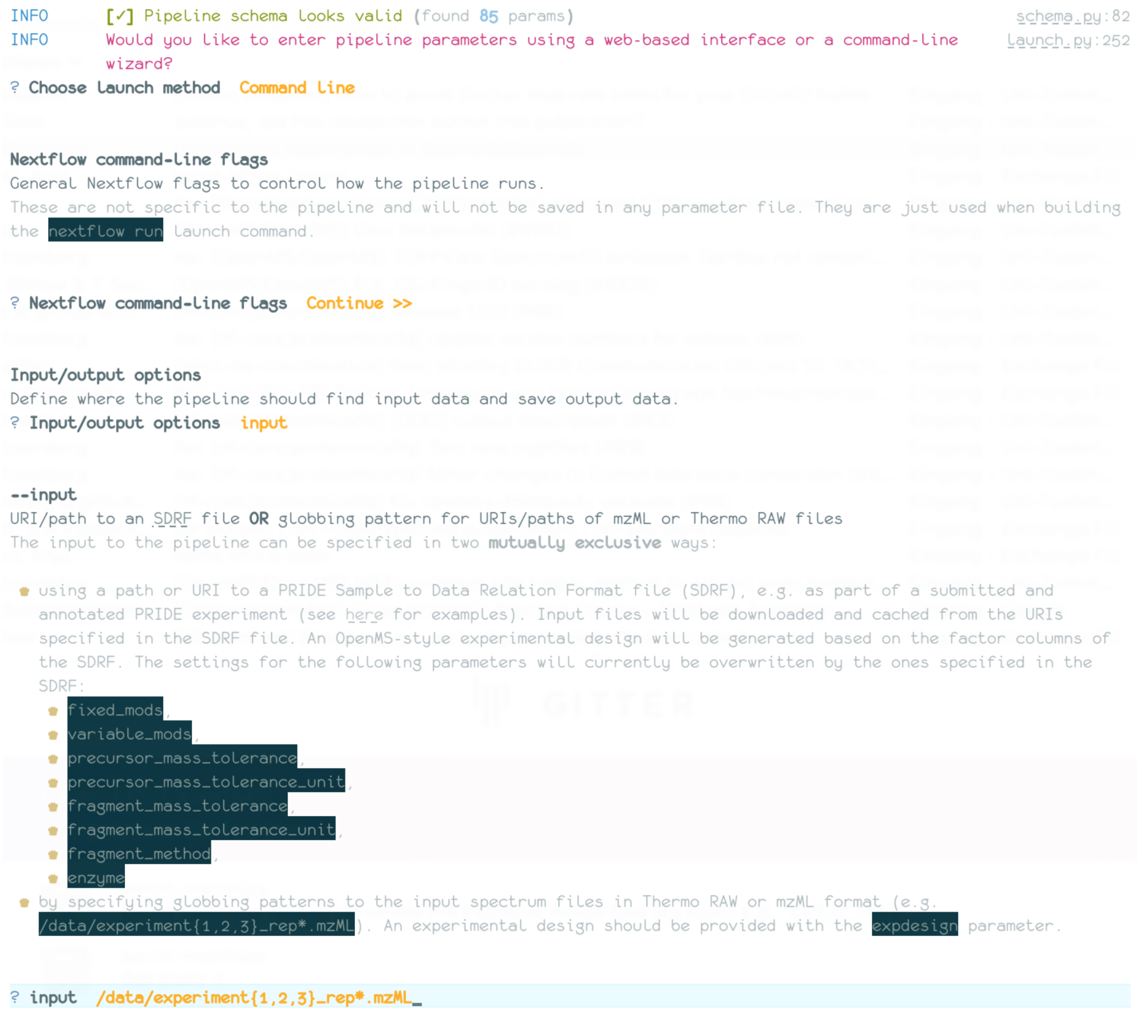
```
nf-core launch proteomicslfq
```

Parameterization then takes place category by category, either on a guided command-line (Figure 5.8) or on a (offline) web-interface with detailed help texts (Figure 5.9). Configuration of the connection to a high-performance computing backend is done via specifying a profile, either from a set of profiles for nextflow’s supported executors that are provided by the nf-core community or one that the users provide themselves. A large collection of profiles for public clouds as well as cluster systems of research institutions is already available in the nf-core repository[i]. The pipeline was successfully tested on in the following environments:

- locally on a MacBook Pro 2014,
- in conjunction with a SLURM[172] cluster of virtualized nodes on the de.NBI cloud,
- a single high-performance server with a simple unorchestrated docker daemon,
- a single virtual machine with a simple docker daemon,
- PRIDE’s large LSF cluster,
- as well as on Amazon’s cloud on instances of nf-core’s AWS subscription.

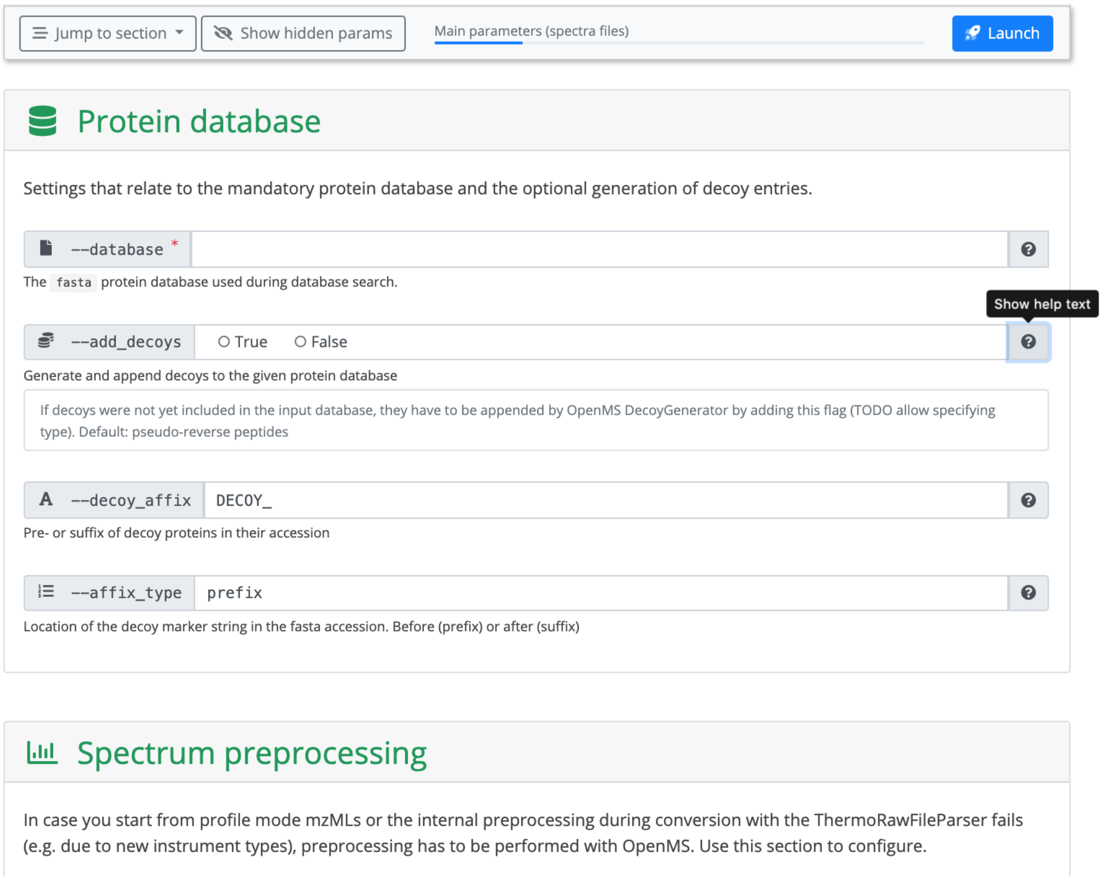
[i] https://github.com/bigbio/proteomics-metadata-standard/tree/master/annotated-projects

[i] https://github.com/nf-core/configs



```
INFO     [✓] Pipeline schema looks valid (found 85 params)                             schema.py:82
INFO     Would you like to enter pipeline parameters using a web-based interface or a command-line   launch.py:252
         wizard?
? Choose launch method  Command line


Nextflow command-line flags
General Nextflow flags to control how the pipeline runs.
These are not specific to the pipeline and will not be saved in any parameter file. They are just used when building
the nextflow run launch command.


? Nextflow command-line flags  Continue >>


Input/output options
Define where the pipeline should find input data and save output data.
? Input/output options  input


--input
URI/path to an SDRF file OR globbing pattern for URIs/paths of mzML or Thermo RAW files
The input to the pipeline can be specified in two mutually exclusive ways:

 • using a path or URI to a PRIDE Sample to Data Relation Format file (SDRF), e.g. as part of a submitted and
   annotated PRIDE experiment (see here for examples). Input files will be downloaded and cached from the URIs
   specified in the SDRF file. An OpenMS-style experimental design will be generated based on the factor columns of
   the SDRF. The settings for the following parameters will currently be overwritten by the ones specified in the
   SDRF:
    • fixed_mods,
    • variable_mods,
    • precursor_mass_tolerance,
    • precursor_mass_tolerance_unit,
    • fragment_mass_tolerance,
    • fragment_mass_tolerance_unit,
    • fragment_method,
    • enzyme
 • by specifying globbing patterns to the input spectrum files in Thermo RAW or mzML format (e.g.
   /data/experiment{1,2,3}_rep*.mzML). An experimental design should be provided with the expdesign parameter.


? input  /data/experiment{1,2,3}_rep*.mzML_
```

Figure 5.8: Guided command-line interface for the configuration and execution of the proteomicslfq pipeline through the `nf-core launch` console command.



Protein database

Settings that relate to the mandatory protein database and the optional generation of decoy entries.

--database *

The fasta protein database used during database search.

--add_decoys ○ True ○ False

Generate and append decoys to the given protein database

If decoys were not yet included in the input database, they have to be appended by OpenMS DecoyGenerator by adding this flag (TODO allow specifying type). Default: pseudo-reverse peptides

--decoy_affix DECOY_

Pre- or suffix of decoy proteins in their accession

--affix_type prefix

Location of the decoy marker string in the fasta accession. Before (prefix) or after (suffix)

Spectrum preprocessing

In case you start from profile mode mzMLs or the internal preprocessing during conversion with the ThermoRawFileParser fails (e.g. due to new instrument types), preprocessing has to be performed with OpenMS. Use this section to configure.

Figure 5.9: Web interface for the configuration of the proteomicslfq pipeline through the “web-based interface” option of the `nf-core launch` console command or directly via the online resource `nf-co.re/launch/proteomicslfq`. When using the command line, generated parameter configurations can be run immediately after configuration. Via the website, it can be either forwarded to the web-based nf-tower (account required) to be run on a registered compute environment or a token can be copied for manual triggering through the command line on a computer with a nextflow installation.

Alternatively, if remote access to the nextflow-enabled HPC environment is configured and registered in a user's nf-tower account, the workflow can be completely launched, configured and monitored from a browser. Parameterization takes place via a simple json file that is automatically transferred to nf-tower after filling out the online version of the configuration dialog on `nf-co.re/launch` and choosing "Launch using Nextflow Tower". nf-tower additionally provides a CLI and offers capabilities for basic data management and workflow chaining.

Part of the output of the pipeline are also quality control reports (as HTML or PDF) of important metrics such as the distribution of precursor mass errors, fragment spectra identification rates, heatmaps of peptide intensities, and upset plots of identified proteins. We are leveraging the strengths of both, the established PTXQC[11] and a self-developed plugin for the nf-core-native multiQC[41] called pMultiQC.

5.4 Conclusion and Outlook

With the nf-core ProteomicsLFQ pipeline, we created a robust and easy-to-use workflow for a large variety of label-free proteomics datasets and compute environments. The pipeline features some of the latest algorithms for peptide identification, peptide rescoring, confidence estimation, consensus scoring, PTM localization, protein inference, and protein quantification. Additionally, it provides convenient reports and state-of-the-art algorithms for quality control and statistical evaluation.

The pipeline is especially well-suited for the one-command re-analysis of datasets from the largest proteomics database PRIDE through its support of PRIDE's experiment description format and automatic staging mechanisms. With this, we plan to extend our current efforts to re-analyze a large part of PRIDE's compatible datasets for easy exploration and summarization of this immense data volume.

Taking advantage of the pipeline's open-source nature and its membership in nf-core, it will be well-maintained and regularly tested through efforts of both the OpenMS developers and the nf-core community (as well as hopefully other future contributors).

Although the current focus was a nextflow implementation due its advantages in large-scale distributed analyses and continuous testing, versions of the pipeline can also be found for KNIME [i] (with extended interactive downstream visualization capabilities) and Galaxy (with its guided tutorials and its easily accessible public server).

Potential algorithmic improvements that can already be identified today would mainly revolve around the handling of the large input data in the identification-based

[i]`https://hub.knime.com/openms-team/spaces/Blog%20workflows/latest/Protein%20label-free%20quantification~Q8TgAl_OAuerfhES`

quantification. We envision that querying identifications and potentially also spectra from indexed out-of-memory databases will allow for a more fine-grained control over which data is needed at which point in the algorithm and thereby significantly reduce the memory footprint.

Regarding the accuracy of the results, although the pipeline offers different parameters to balance the sensitivity and specificity trade-off, it would benefit from recently emerging target-decoy approaches for quantified features (in particular unidentified ones only included through matching between runs). For some quantities in the low concentrations of the UPS dataset it is not clear if a feature from a different entity was picked or if the traces in the feature were so close to noise that trace fitting led to longer than needed feature boundaries and additional noise peaks were summed in- and outside of the actual elution range. Both would explain the underestimation of the true fold changes for spike-ins between concentrations under 2500 amol.

Additions could be made through providing alternatives to existing steps (e.g., adding other supported search engines or PTM localization tools). Also, new steps could be introduced to support analysis techniques like a cascaded search or open modification searches. We are also hoping for future contributions in regard to quality control metrics and their visualization. While PTXQC provides a set of metrics based on the final mzTab it would be beneficial to report on the quality of intermediate steps as well since not all the information is propagated. An integration of more metrics into the nf-core quasi-native multiQC tool[41], called pmultiQC is currently under development.

We are confident that future pipelines will benefit from the developments and the general skeleton of ProteomicsLFQ. For example, a workflow for protein quantification with isobaric labels is already in development, with the long-term goal of providing a unified quantification pipeline for the most commonly used quantification strategies in proteomics.

Lastly, more rigorous benchmarking against other pipelines and tools is currently performed in an ELIXIR community effort.

Chapter 6

Conclusion

In recent years, the scale of proteomics experiments increased significantly. Single experiments get larger (meta-proteomics, cite one experiment) and multiple experiments are more often combined to create an overview of the complete protein landscape of an organism[171]. Simultaneously, the number of datasets in public proteomics databases has increased rapidly. It became evident that automated analysis pipelines with scalable algorithms are needed to keep up with the growth of data. It would not only enable quick analysis of large sample cohorts in core facilities but also allows a quality control and uniform summarization of the information in public datasets to be used for open (e.g., web-based) exploration or the generation of machine learning training data.

Hence, we evaluated OpenMS, a well-known and commonly used framework for computational mass spectrometry data analysis for its capabilities to perform the abovementioned task. Firstly, it was found that it lacked data structures and an efficient algorithm for large-scale protein inference. Therefore, after surveying the landscape of inference algorithms[163], a Bayesian approach was chosen to be pursued. In this thesis we therefore refined a state-of-the-art max-product inference procedure for large converging connections as they are found in the protein inference problem. A more stable approximate max-convolution method was developed that leverages curvature information to extrapolate the infinite norm of vectors while still retaining invertibility. Advances in basic research like this did not only allow the stable usage of convolution trees[141] for approximate max-product inference in Bayesian protein inference tasks but a large range of other problems of similar structure (e.g., inference in hidden Markov models in finance[118]) or even seemingly unrelated but fundamental problems (e.g., all-pairs-shortest-path problems or efficiently sorting all pairs of sums $x_i + y_j$ based on the input vectors $\vec{x}$ and $\vec{y}$[139]).

As a second contribution, based on this fast max-convolution method and related advances in inference on discrete Bayesian networks[141], an inference library was co-developed and a follow-up, stable version integrated into OpenMS. It features an approximate inference algorithm called “loopy belief propagation”[99], that allows us to efficiently apply our previously researched algorithms on Bayesian networks with cycles as in all but the most simple bipartite protein-peptide networks. All developments were integrated into the command line tool EPIFANY that apart from the largely improved scalability compared to most existing methods, also offers some novelties in modeling the problem for increased calibration (e.g., through penalizing multiple parent proteins) as well as more input and output options (such as user-defined protein priors and posterior probabilities for peptides). Lastly, since the quality of inference results depends heavily on the correct error estimation of peptide search engine results and since the ultimate goal is often the quantification of proteins in complex experimental designs, we constructed an easy-to-use but scalable pipeline for automated quantitative analyses of label-free shotgun proteomics experiments, called ProteomicsLFQ[119]. It builds on the developments of protein-level algorithms and data structures in OpenMS, that were introduced with EPIFANY but also integrates improvements and additions for pre-processing, database search, error estimation, and discovery of quantifiable features. Inference is now tightly coupled to quantification in a single tool that handles the logic for complex experimental designs, such that the structure of the overall workflow can be reused for the vast majority of label-free experiments. The experimental design, that can be read from SDRF[114] files for one-command re-analyses of PRIDE datasets, now enables a 100% validated mzTab[49] file as output, ready to be (re-)uploaded to PRIDE. We showed on ground truth and large-scale datasets, that both the accuracy and scalability of the pipeline is state-of-the-art and simple configurations for many HPC compute infrastructures makes it a viable candidate for core facilities in mass spectrometry research. A large-scale re-analysis of recently annotated PRIDE datasets is ongoing. ProteomicsLFQ is planned to be a basis for new services in PRIDE to query expression values of proteins across the whole database. Also, slim web-based user-interfaces and automated reports for both quality control and statistical analysis enables a wide use for smaller labs without bioinformaticians. All tools used in the pipeline are frozen in their version and packaged in containers such that results generated with the pipeline are completely reproducible. Open-source development on GitHub with multiple contributors and the continuous testing of the OpenMS tools as well as the complete pipeline will additionally ensure long-term stability and maintenance through updates and additions.

In the ever-changing field of mass spectrometry-based proteomics it is important for researchers to be able to rely on flexible, robust and sustainable software for the analysis of the growing amount of data. In this thesis we not only contributed such software but also created a start-to-end workflow that fulfills these requirements. We are confident that the outcome of our endeavors will help shape the future of large-scale automated analysis of bottom-up proteomics datasets of various designs and sizes to yield valuable insights for numerous fields of life sciences.

Bibliography

[1] Enis Afgan et al. “The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update”. In: *Nucleic Acids Research* 46.W1 (July 2018), W537–W544. ISSN: 13624962. DOI: 10.1093/nar/gky379 (cit. on pp. 39, 107).

[2] John Mark Agosta. “The Structure of Bayes Networks for Visual Recognition”. In: *Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence*. UAI ’88. Amsterdam, The Netherlands: North-Holland Publishing Co., 1990, pp. 397–406. ISBN: 0-444-88650-8 (cit. on p. 36).

[3] Zeeshan Ahmed, Saman Zeeshan, and Thomas Dandekar. “Developing sustainable software solutions for bioinformatics by the “Butterfly” Paradigm”. In: *F1000Research* 3 (Aug. 2014). ISSN: 1759796X. DOI: 10.12688/f1000research.3681.2. URL: /pmc/articles/PMC4215756/%20/pmc/articles/PMC4215756/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4215756/ (cit. on p. 109).

[4] P. Alves et al. “Advancement in protein inference from shotgun proteomics using peptide detectability”. In: *Proceedings of the Pacific Symposium on Biocomputing*. Singapore: World Scientific, 2007, pp. 409–420 (cit. on p. 27).

[5] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. “Complexity of finding embeddings in a k-tree”. In: *SIAM Journal on Algebraic Discrete Methods* 8.2 (1987), pp. 277–284 (cit. on p. 33).

[6] Enrique Audain et al. “In-depth analysis of protein inference algorithms using multiple search engines and well-defined metrics”. In: *Journal of Proteomics* 150 (Jan. 2017), pp. 170–182. ISSN: 1874-3919. DOI: 10.1016/J.JPROT.2016.08.002. URL: https://www.sciencedirect.com/science/article/pii/S187439191630344X?via%7B%5C%%7D3Dihub (cit. on p. 90).

[7] V. Bafna and N. Edwards. “SCOPE: a probabilistic model for scoring tandem mass spectra against a peptide database”. In: *Bioinformatics* 17.Suppl 1 (June 2001), S13–S21. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/17.suppl_1.S13. URL: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/17.suppl%7B%5C_%7D1.S13 (cit. on p. 12).

[8] A Bairoch and R Apweiler. “The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000.” In: *Nucleic acids research* 28.1 (Jan. 2000), pp. 45–48. ISSN: 0305-1048. URL: http://www.ncbi.nlm.nih.gov/pubmed/10592178%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC102476 (cit. on p. 89).

[9] Y. Benjamini and Y. Hochberg. “Controlling the false discovery rate: a practical and powerful approach to multiple testing”. In: *Journal of the Royal Statistical Society B* 57 (1995), pp. 289–300 (cit. on p. 17).

[10] C. Berrou, A. Glavieux, and P. Thitimajshima. “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1”. In: *Proceedings of ICC ’93 - IEEE International Conference on Communications*. Vol. 2. IEEE, 1993, pp. 1064–1070. ISBN: 0-7803-0950-2. DOI: 10.1109/ICC.1993.397441. URL: http://ieeexplore.ieee.org/document/397441/ (cit. on p. 81).

[11] Chris Bielow, Guido Mastrobuoni, and Stefan Kempa. “Proteomics Quality Control: Quality Control Software for MaxQuant Results”. In: *Journal of Proteome Research* 15.3 (Mar. 2016), pp. 777–787. ISSN: 1535-3893. DOI: 10.1021/acs.jproteome.5b00780. URL: https://pubs.acs.org/doi/10.1021/acs.jproteome.5b00780 (cit. on pp. 115, 129).

[12] Marc Boyer, Guillaume Dufour, and Luca Santinelli. “Continuity for network calculus”. In: *Proceedings of the 21st International conference on Real-Time Networks and Systems - RTNS ’13*. New York, New York, USA: ACM Press, Oct. 2013, p. 235. ISBN: 9781450320580. DOI: 10.1145/2516821.2516840. URL: http://dl.acm.org/citation.cfm?id=2516821.2516840 (cit. on p. 42).

[13] David Bremner et al. “Necklaces, convolutions, and X+Y”. In: *Algorithmica* 69.2 (Dec. 2014), pp. 294–314. ISSN: 14320541. DOI: 10.1007/s00453-012-9734-3. arXiv: 1212.4771. URL: https://link.springer.com/article/10.1007/s00453-012-9734-3 (cit. on p. 42).

[14] Michael Bussieck et al. “Fast algorithms for the maximum convolution problem”. In: *Operations Research Letters* 15.3 (Apr. 1994), pp. 133–141. ISSN: 01676377. DOI: 10.1016/0167-6377(94)90048-5 (cit. on pp. 42, 70, 71, 76).

[15] Joseph E. Campana. “Time-of-Flight Mass Spectrometry: a Historical Overview”. In: *Instrumentation Science & Technology* 16.1 (Jan. 1987), pp. 1–14. ISSN: 1073-9149. DOI: 10.1080/10739148708543625. URL: http://www.tandfonline.com/doi/abs/10.1080/10739148708543625 (cit. on p. 7).

[16] Steven Carr et al. “The Need for Guidelines in Publication of Peptide and Protein Identification Data”. In: *Molecular & Cellular Proteomics* 3.6 (2004), pp. 531–533. DOI: 10.1074/mcp.T400006-MCP200. eprint: http://www.mcponline.org/content/3/6/531.full.pdf+html. URL: http://www.mcponline.org/content/3/6/531.short (cit. on p. 23).

[17] Matthew C Chambers et al. “A cross-platform toolkit for mass spectrometry and proteomics”. In: *Nature Biotechnology* 30 (Oct. 2012), pp. 918–920 (cit. on p. 88).

[18] Ting Chen et al. “A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry”. In: *Journal of Computational Biology* 8.3 (2001), pp. 325–337. ISSN: 10665277. DOI: 10.1089/10665270152530872. arXiv: 0101016 [cs]. URL: https://pubmed.ncbi.nlm.nih.gov/11535179/ (cit. on p. 12).

[19] Hyungwon Choi and Alexey I. Nesvizhskii. “Semisupervised model-based validation of peptide identifications in mass spectrometry-based proteomics”. In: *Journal of Proteome Research* 7.1 (Jan. 2008), pp. 254–265. ISSN: 15353893. DOI: 10.1021/pr070542g. URL: https://pubs.acs.org/sharingguidelines (cit. on p. 16).

[20] Meena Choi et al. "MSstats: an R package for statistical analysis of quantitative mass spectrometry-based proteomic experiments". In: *Bioinformatics* 30.17 (Sept. 2014), pp. 2524–2526. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu305. URL: https://academic.oup.com/bioinformatics/bioinformatics/article/2748156/MSstats: (cit. on p. 115).

[21] Melvin B Comisarow and Alan G Marshall. "Fourier transform ion cyclotron resonance spectroscopy". In: *Chemical physics letters* 25.2 (1974), pp. 282–283 (cit. on p. 16).

[22] Melvin B. Comisarow and Alan G. Marshall. "Fourier transform ion cyclotron resonance spectroscopy". In: *Chemical Physics Letters* 25.2 (Mar. 1974), pp. 282–283. ISSN: 00092614. DOI: 10.1016/0009-2614(74)89137-2. URL: https://linkinghub.elsevier.com/retrieve/pii/0009261474891372 (cit. on p. 7).

[23] Philip D. Compton et al. "On the scalability and requirements of whole protein mass spectrometry". In: *Analytical Chemistry* 83.17 (Sept. 2011), pp. 6868–6874. ISSN: 00032700. DOI: 10.1021/ac2010795. URL: https://pubmed.ncbi.nlm.nih.gov/21744800/ (cit. on pp. 6, 21).

[24] Gregory F Cooper. "The computational complexity of probabilistic inference using Bayesian belief networks". In: *Artificial Intelligence* 42.2 (1990), pp. 393–405 (cit. on p. 32).

[25] John S. Cottrell. "Protein identification using MS/MS data". In: *Journal of Proteomics* 74.10 (Sept. 2011), pp. 1842–1851. ISSN: 1874-3919. DOI: 10.1016/J.JPROT.2011.05.014. URL: https://www.sciencedirect.com/science/article/pii/S1874391911002053 (cit. on p. 12).

[26] Jürgen Cox and Matthias Mann. "MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification". In: *Nature Biotechnology* 26.12 (Dec. 2008), pp. 1367–1372. ISSN: 10870156. DOI: 10.1038/nbt.1511. URL: https://www.nature.com/articles/nbt.1511 (cit. on p. 107).

[27] Jürgen Cox and Matthias Mann. "MaxQuant enables high peptide identification rates, individualized ppb-range mass accuracies and proteome-wide protein quantification". In: *Nature biotechnology* 26.12 (2008), pp. 1367–1372 (cit. on pp. 3, 118).

[28] Robertson Craig and Ronald C Beavis. "TANDEM: matching proteins with tandem mass spectra." In: *Bioinformatics (Oxford, England)* 20.9 (June 2004), pp. 1466–7. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bth092. URL: http://www.ncbi.nlm.nih.gov/pubmed/14976030 (cit. on pp. 12, 13, 16).

[29] Felipe da Veiga Leprevost et al. "BioContainers: an open-source and community-driven framework for software standardization". In: *Bioinformatics* 33.16 (Aug. 2017). Ed. by Alfonso Valencia, pp. 2580–2582. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx192. URL: https://academic.oup.com/bioinformatics/article/33/16/2580/3096437 (cit. on p. 116).

[30] Sven Degroeve, Lennart Martens, and Igor Jurisica. "MS2PIP: A tool for MS/MS peak intensity prediction". In: *Bioinformatics* 29.24 (Dec. 2013), pp. 3199–3203. ISSN: 14602059. DOI: 10.1093/bioinformatics/btt544. URL: https://pubmed.ncbi.nlm.nih.gov/24078703/ (cit. on p. 13).

[31] Eric W. Deutsch et al. "Trans-Proteomic Pipeline, a standardized data processing pipeline for large-scale reproducible proteomics informatics". In: *PROTEOMICS - Clinical Applications* 9.7-8 (Aug. 2015), pp. 745–754. ISSN: 18628346. DOI: 10.1002/prca.201400164. URL: http://www.ncbi.nlm.nih.gov/pubmed/25631240%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4506239%20http://doi.wiley.com/10.1002/prca.201400164 (cit. on p. 91).

[32] Paolo Di Tommaso et al. "Nextflow enables reproducible computational workflows". In: *Nature Biotechnology* 35.4 (Apr. 2017), pp. 316–319. ISSN: 1087-0156. DOI: 10.1038/nbt.3820. URL: http://www.nature.com/articles/nbt.3820 (cit. on pp. 3, 39, 108).

[33] P. Domingos and M. Pazzani. "Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier". In: *Proceedings of the International Conference on Machine Learning*. 1996 (cit. on p. 25).

[34] Viktoria Dorfer et al. "MS Amanda, a universal identification algorithm optimized for high accuracy tandem mass spectra". In: *Journal of Proteome Research* 13.8 (2014), pp. 3679–3684 (cit. on p. 16).

[35] Olive Jean Dunn. "Multiple comparisons among means". In: *Journal of the American Statistical Association* 56.293 (1961), pp. 52–64 (cit. on p. 17).

[36] Arthur Stanley Eddington. *Mathematical Theory of Relativity*. London: Cambridge University Press, 1923. ISBN: 0828402787 (cit. on p. 55).

[37] J. E. Elias and S. P. Gygi. "Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry". In: *Nature Methods* 4.3 (2007), pp. 207–214 (cit. on p. 19).

[38] Gal Elidan, Ian McGraw, and Daphne Koller. "Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing". In: *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence 2006*. June 2006. arXiv: 1206.6837. URL: http://arxiv.org/abs/1206.6837 (cit. on p. 85).

[39] J K Eng, A L McCormack, and J R Yates. "An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database." In: *Journal of the American Society for Mass Spectrometry* 5.11 (Nov. 1994), pp. 976–89. ISSN: 1044-0305. DOI: 10.1016/1044-0305(94)80016-2. URL: http://www.ncbi.nlm.nih.gov/pubmed/24226387 (cit. on pp. 12, 13).

[40] Jimmy K. Eng, Tahmina A. Jahan, and Michael R. Hoopmann. "Comet: An open-source MS/MS sequence database search tool". In: *PROTEOMICS* 13.1 (Jan. 2013), pp. 22–24. ISSN: 16159853. DOI: 10.1002/pmic.201200439. URL: http://doi.wiley.com/10.1002/pmic.201200439 (cit. on pp. 12, 13, 16, 88).

[41] Philip Ewels et al. "MultiQC: summarize analysis results for multiple tools and samples in a single report". In: *Bioinformatics* 32.19 (Oct. 2016), pp. 3047–3048. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw354. URL: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btw354 (cit. on pp. 129, 130).

[42] Philip A. Ewels et al. *The nf-core framework for community-curated bioinformatics pipelines*. Mar. 2020. DOI: `10.1038/s41587-020-0439-x` (cit. on pp. 3, 109, 116).

[43] Damian Fermin et al. "LuciPHOr2: site localization of generic post-translational modifications from tandem mass spectrometry data". In: *Bioinformatics* 31.7 (Apr. 2015), pp. 1141–1143. ISSN: 1460-2059. DOI: `10.1093/bioinformatics/btu788`. URL: `https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btu788` (cit. on p. 114).

[44] Christian K Frese et al. "Toward full peptide sequence coverage by dual fragmentation combining electron-transfer and higher-energy collision dissociation tandem mass spectrometry". In: *Analytical chemistry* 84.22 (2012), pp. 9668–9673 (cit. on p. 8).

[45] Vincent A Fusaro et al. "Prediction of high-responding peptides for targeted protein assays by mass spectrometry." In: *Nature Biotechnology* 27.2 (2009), pp. 190–198 (cit. on pp. 27, 30).

[46] S. Gerster et al. "Protein and gene model inference based on statistical modeling in k-partite graphs". In: *Proceedings of the National Academy of Sciences* 107.27 (2010), pp. 12101–12106 (cit. on p. 31).

[47] Siegfried Gessulat et al. "Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning". In: *Nature Methods* 16.6 (June 2019), pp. 509–518. ISSN: 15487105. DOI: `10.1038/s41592-019-0426-7`. URL: `https://www.nature.com/articles/s41592-019-0426-7` (cit. on p. 13).

[48] C. A. Goble et al. "myExperiment: a repository and social network for the sharing of bioinformatics workflows". In: *Nucleic Acids Research* 38.Web Server (July 2010), W677–W682. ISSN: 0305-1048. DOI: `10.1093/nar/gkq429`. URL: `https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkq429` (cit. on p. 109).

[49] Johannes Griss et al. "The mzTab data exchange format: communicating mass-spectrometry-based proteomics and metabolomics experimental results to a wider audience." In: *Molecular & cellular proteomics : MCP* 13.10 (Oct. 2014), pp. 2765–75. ISSN: 1535-9484. DOI: `10.1074/mcp.O113.036681`. URL: `http://www.ncbi.nlm.nih.gov/pubmed/24980485%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4189001` (cit. on pp. 112, 132).

[50] Björn Grüning et al. "Bioconda: sustainable and comprehensive software distribution for the life sciences". In: *Nature Methods* 15.7 (July 2018), pp. 475–476. ISSN: 1548-7091. DOI: `10.1038/s41592-018-0046-7`. URL: `http://www.nature.com/articles/s41592-018-0046-7` (cit. on pp. 3, 39, 116).

[51] Eli. Grushka. "Characterization of exponentially modified Gaussian peaks in chromatography". In: *Analytical Chemistry* 44.11 (Sept. 1972), pp. 1733–1738. ISSN: 0003-2700. DOI: `10.1021/ac60319a011`. URL: `https://pubs.acs.org/doi/abs/10.1021/ac60319a011` (cit. on p. 11).

[52] N. Gupta and P. Pevzner. "False discovery rates of protein identifications: a strike against the two-peptide rule". In: *Journal of Proteome Research* 8.9 (2009), pp. 4173–4181 (cit. on p. 25).

[53] John T. Halloran, Jeff A. Bilmes, and William S. Noble. “Dynamic Bayesian network for accurate detection of peptides from tandem mass spectra”. In: *Journal of Proteome Research* 15.8 (Aug. 2016), pp. 2749–2759. ISSN: 15353907. DOI: 10.1021/acs.jproteome.6b00290. URL: https://pubmed.ncbi.nlm.nih.gov/27397138/ (cit. on p. 12).

[54] Zengyou He, Can Yang, and Weichuan Yu. “A Partial Set Covering Model for Protein Mixture Identification Using Mass Spectrometry Data”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8.2 (2011), pp. 368–380. ISSN: 1545-5963. DOI: http://doi.ieeecomputersociety.org/10.1109/TCBB.2009.54 (cit. on p. 26).

[55] C S Ho et al. “Electrospray ionisation mass spectrometry: principles and clinical applications.” In: *The Clinical Biochemist Reviews* 24.1 (2003), pp. 3–12. ISSN: 0159-8090. URL: http://www.ncbi.nlm.nih.gov/pubmed/18568044%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC1853331 (cit. on p. 7).

[56] Edmond de Hoffmann and Vincent Stroobant. *Mass Spectrometry: Principles and Applications*. Wiley, 2003, p. 65. ISBN: 978-0-471-48566-7 (cit. on p. 7).

[57] Roger A Horn and Charles R Johnson. *Matrix Analysis*. 2nd Editio. Cambridge: Cambridge University Press, 2012. ISBN: 9781139020411. DOI: 10.1017/CBO9781139020411. URL: http://ebooks.cambridge.org/ref/id/CBO9781139020411 (cit. on p. 155).

[58] Qizhi Hu et al. “The Orbitrap: a new mass spectrometer”. In: *Journal of mass spectrometry* 40.4 (2005), pp. 430–443 (cit. on p. 16).

[59] Ting Huang and Zengyou He. “A linear programming model for protein inference problem in shotgun proteomics”. In: *Bioinformatics* 28.22 (2012), pp. 2956–2962 (cit. on p. 27).

[60] Christopher Hughes, Bin Ma, and Gilles A. Lajoie. “De novo sequencing methods in proteomics.” In: *Methods in molecular biology* 604 (2010), pp. 105–121. ISSN: 19406029. DOI: 10.1007/978-1-60761-444-9_8. URL: https://pubmed.ncbi.nlm.nih.gov/20013367/ (cit. on p. 12).

[61] Tommi S Jaakkola and Michael I Jordan. “Bayesian parameter estimation via variational methods”. In: *Statistics and Computing* 10.1 (2000), pp. 25–37 (cit. on p. 33).

[62] Jensen, SL Lauritzen, and KG Olesen. “Bayesian updating in causal probabilistic networks by local computations”. In: *Computational Statistics Quaterly* 4 (1990), pp. 269–282 (cit. on pp. 33, 34, 85, 168).

[63] Marko Jovanovic et al. “Dynamic profiling of the protein life cycle in response to pathogens”. In: *Science* 347.6226 (Mar. 2015). ISSN: 10959203. DOI: 10.1126/science.1259038. URL: https://pubmed.ncbi.nlm.nih.gov/25745177/ (cit. on p. 5).

[64] E. Kail et al. “Dynamic workflow support in gUSE”. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, May 2014, pp. 354–359. ISBN: 978-953-233-077-9. DOI: 10.1109/MIPRO.2014.6859590. URL: http://ieeexplore.ieee.org/document/6859590/ (cit. on p. 39).

[65] L. Käll, M. J. MacCoss, and W. S. Noble. “A method that both predicts and incorporates retention time into a peptide-spectrum match scoring function”. In preparation. 2009 (cit. on p. 16).

[66] L. Käll et al. “Assigning significance to peptides identified by tandem mass spectrometry using decoy databases”. In: *Journal of Proteome Research* 7.1 (2008), pp. 29–34 (cit. on p. 19).

[67] Lukas Käll et al. “Semi-supervised learning for peptide identification from shotgun proteomics datasets”. In: *Nature Methods* 4.11 (Nov. 2007), pp. 923–925. ISSN: 1548-7091. DOI: 10.1038/nmeth1113. URL: http://www.ncbi.nlm.nih.gov/pubmed/17952086%20http://www.nature.com/doifinder/10.1038/nmeth1113 (cit. on p. 16).

[68] R. M. Karp. “Reducibility Among Combinatorial Problems”. In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. Plenum Press, 1972, pp. 85–103 (cit. on p. 26).

[69] Uri Keich, Attila Kertesz-Farkas, and William Stafford Noble. “Improved False Discovery Rate Estimation Procedure for Shotgun Proteomics”. In: *Journal of Proteome Research* 14.8 (2015). PMID: 26152888, pp. 3148–3161. DOI: 10.1021/acs.jproteome.5b00081 (cit. on p. 19).

[70] Uri Keich and William Stafford Noble. “On the importance of well-calibrated scores for identifying shotgun proteomics spectra”. In: *Journal of Proteome Research* 14.2 (Feb. 2015), pp. 1147–1160. ISSN: 15353907. DOI: 10.1021/pr5010983. URL: https://pubs.acs.org/sharingguidelines (cit. on p. 20).

[71] Andrew Keller et al. “Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search”. In: *Analytical Chemistry* 74.20 (Oct. 2002), pp. 5383–5392. ISSN: 00032700. DOI: 10.1021/ac025747h. URL: https://pubmed.ncbi.nlm.nih.gov/12403597/ (cit. on p. 16).

[72] Sangtae Kim, Nitin Gupta, and Pavel A Pevzner. “Spectral probabilities and generating functions of tandem mass spectra: a strike against decoy databases.” In: *Journal of proteome research* 7.8 (Aug. 2008), pp. 3354–63. ISSN: 1535-3893. DOI: 10.1021/pr8001244. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2689316%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract (cit. on pp. 12, 13, 16, 20, 89).

[73] A. A. Klammer et al. “Improving tandem mass spectrum identification using peptide retention time prediction across diverse chromatography conditions”. In: *Analytical Chemistry* 79.16 (2007), pp. 6111–8 (cit. on p. 16).

[74] Aaron A. Klammer, Christopher Y. Park, and William Stafford Noble. “Statistical calibration of the SEQUEST Xcorr function”. In: *Journal of Proteome Research* 8.4 (Apr. 2009), pp. 2106–2113. ISSN: 15353893. DOI: 10.1021/pr8011107. URL: https://pubs.acs.org/sharingguidelines (cit. on p. 20).

[75] Andy T. Kong et al. “MSFragger: Ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics”. In: *Nature Methods* 14.5 (Apr. 2017), pp. 513–520. ISSN: 15487105. DOI: 10.1038/nmeth.4256. URL: https://www.nature.com/articles/nmeth.4256 (cit. on p. 13).

[76] U. K. Laemmli. “Cleavage of Structural Proteins during the Assembly of the Head of Bacteriophage T4”. In: *Nature* 227.5259 (Aug. 1970), pp. 680–685. ISSN: 0028-0836. DOI: 10.1038/227680a0. URL: http://www.nature.com/articles/227680a0 (cit. on p. 7).

[77] Eva Lange et al. “A geometric approach for the alignment of liquid chromatography—mass spectrometry data”. In: *Bioinformatics* 23.13 (July 2007), pp. i273–i281. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btm209. URL: https://academic.oup.com/bioinformatics/article/23/13/i273/233877 (cit. on p. 11).

[78] Vasilica Lepar and Prakash P Shenoy. “A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions”. In: *arXiv preprint arXiv:1301.7394* (2013) (cit. on pp. 33–35).

[79] Lev I. Levitsky et al. “Unbiased False Discovery Rate Estimation for Shotgun Proteomics Based on the Target-Decoy Approach”. In: *Journal of Proteome Research* 16.2 (Feb. 2017), pp. 393–397. ISSN: 1535-3893. DOI: 10.1021/acs.jproteome.6b00144. URL: http://pubs.acs.org/doi/10.1021/acs.jproteome.6b00144 (cit. on pp. 90, 114).

[80] Q. Li, M. J. MacCoss, and M. Stephens. “A nested mixture model for protein identification using mass spectrometry”. In: *Annals of Applied Sciences* 4.2 (2010), pp. 962–987 (cit. on p. 29).

[81] Y. F. Li and P. Radivojac. “Computational approaches to protein inference in shotgun proteomics”. In: *BMC bioinformatics* 13.Suppl 16 (2012), S4 (cit. on p. 28).

[82] Y. F. Li et al. “A Bayesian approach to protein inference problem in shotgun proteomics”. In: *Proceedings of the Twelfth Annual International Conference on Computational Molecular Biology*. Ed. by M. Vingron and L. Wong. Vol. 12. Lecture Notes in Bioinformatics. Berlin, Germany: Springer, 2008, pp. 167–180 (cit. on pp. 30, 81).

[83] Yong Fuga Li et al. “A Bayesian approach to protein inference problem in shotgun proteomics.” In: *Journal of Computational Biology* 16.8 (Aug. 2009), pp. 1183–93 (cit. on pp. 30, 81).

[84] Yong Fuga Li et al. “A bayesian approach to protein inference problem in shotgun proteomics.” In: *Journal of computational biology : a journal of computational molecular cell biology* 16.8 (Aug. 2009), pp. 1183–93. ISSN: 1557-8666. DOI: 10.1089/cmb.2009.0018. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2799497%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract (cit. on p. 84).

[85] Nora Linscheid et al. “Quantitative Proteomics of Human Heart Samples Collected In Vivo Reveal the Remodeled Protein Landscape of Dilated Left Atrium Without Atrial Fibrillation”. In: *Molecular & cellular proteomics : MCP* 19.7 (July 2020), pp. 1132–1144. ISSN: 15359484. DOI: 10.1074/mcp.RA119.001878. URL: https://doi.org/10.1074/mcp.RA119.001878 (cit. on p. 124).

[86] Markus List, Peter Ebert, and Felipe Albrecht. “Ten Simple Rules for Developing Usable Software in Computational Biology”. In: *PLOS Computational Biology* 13.1 (Jan. 2017). Ed. by Scott Markel, e1005265. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005265. URL: https://dx.plos.org/10.1371/journal.pcbi.1005265 (cit. on p. 109).

[87] Yansheng Liu, Andreas Beyer, and Ruedi Aebersold. “Leading Edge Review On the Dependency of Cellular Protein Levels on mRNA Abundance”. In: *Cell* 165 (2016), pp. 535–550. DOI: 10.1016/j.cell.2016.03.014. URL: http://dx.doi.org/10.1016/j.cell.2016.03.014 (cit. on p. 5).

[88] Z.-Q. Ma et al. “IDPicker 2.0: Improved Protein Assembly with High Discrimination Peptide Identification Filtering”. In: *Journal of Proteome Research* 8.8 (2009), pp. 3872–3881 (cit. on p. 26).

[89] Alexander Makarov. “Electrostatic Axially Harmonic Orbital Trapping: A High-Performance Technique of Mass Analysis”. In: *Analytical Chemistry* 72.6 (Mar. 2000), pp. 1156–1162. ISSN: 0003-2700. DOI: 10.1021/ac991131p. URL: https://pubs.acs.org/doi/10.1021/ac991131p (cit. on p. 7).

[90] P. Mallick et al. “Computational prediction of proteotypic peptides for quantitative proteomics”. In: *Nature Biotechnology* 25 (2006), pp. 125–131 (cit. on pp. 27, 30).

[91] Lennart Martens et al. “mzML - A community standard for mass spectrometry data”. In: *Molecular and Cellular Proteomics* 10.1 (Jan. 2011). ISSN: 15359476. DOI: 10.1074/mcp.R110.000133. URL: http://www.mcponline.org (cit. on p. 112).

[92] Lennart Martens et al. “PRIDE: the proteomics identifications database”. In: *Proteomics* 5.13 (2005), pp. 3537–3545 (cit. on p. 101).

[93] Vivien Marx. “When computational pipelines go ‘clank’”. In: *Nature Methods* 17.7 (July 2020), pp. 659–662. ISSN: 15487105. DOI: 10.1038/s41592-020-0886-9. URL: www.nature.com/naturemethods (cit. on p. 107).

[94] R.J. McEliece, D.J.C. MacKay, and J. Cheng. “Turbo decoding as an instance of Pearl’s “belief propagation” algorithm”. In: *IEEE Journal on Selected Areas in Communications* 16.2 (1998), pp. 140–152 (cit. on p. 36).

[95] Subina Mehta et al. “Precursor Intensity-Based Label-Free Quantification Software Tools for Proteomic and Multi-Omic Analysis within the Galaxy Platform”. In: *Proteomes* 8.3 (July 2020), p. 15. ISSN: 2227-7382. DOI: 10.3390/proteomes8030015. URL: https://www.mdpi.com/2227-7382/8/3/15 (cit. on p. 109).

[96] I. Molnar and C. Horvath. “Reverse phase chromatography of polar biological substances: separation of catechol compounds by high performance liquid chromatography”. In: *Clinical Chemistry* 22.9 (Sept. 1976), pp. 1497–1502. ISSN: 00099147. DOI: 10.1093/clinchem/22.9.1497. URL: https://europepmc.org/article/MED/8221 (cit. on p. 7).

[97] R. E. Moore, M. K. Young, and T. D. Lee. “Qscore: An algorithm for evaluating SEQUEST database search results”. In: *Journal of the American Society for Mass Spectrometry* 13.4 (2002), pp. 378–386 (cit. on p. 19).

[98] K.P. Murphy, Y. Weiss, and M.I. Jordan. “Loopy belief propagation for approximate inference: An empirical study”. In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. 1999, pp. 467–475 (cit. on pp. 36, 101).

[99] Kevin P Murphy, Yair Weiss, and Michael I Jordan. “Loopy Belief Propagation for Approximate Inference: An Empirical Study”. In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. UAI’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 467–475. ISBN: 1-55860-614-9. URL: http://dl.acm.org/citation.cfm?id=2073796.2073849 (cit. on pp. 81, 84, 85, 131, 175).

[100] Nysret Musliu. *An iterative heuristic algorithm for tree decomposition*. Springer, 2008, pp. 133–150 (cit. on p. 33).

[101] Sven Nahnsen et al. "Probabilistic consensus scoring improves tandem mass spectrometry peptide identification". In: *Journal of Proteome Research* 10.8 (Aug. 2011), pp. 3332–3343. ISSN: 15353893. DOI: 10.1021/pr2002879 (cit. on p. 16).

[102] John A Nelder and Roger Mead. "A simplex method for function minimization". In: *The Computer Journal* 7.4 (1965), pp. 308–313 (cit. on p. 27).

[103] Y. Nesterov and A. Nemirovsky. *Interior-point polynomial methods in convex programming: Theory and applications*. Vol. 13. Studies in Applied Mathematics. Philadelphia, PA: SIAM, 1994 (cit. on pp. 26, 27).

[104] A. I. Nesvizhskii and R. Aebersold. "Interpretation of shotgun proteomic data the protein inference problem". In: *Molecular & Cellular Proteomics* 4.10 (2005), pp. 1419–1440 (cit. on p. 27).

[105] A. I. Nesvizhskii et al. "A statistical model for identifying proteins by tandem mass spectrometry". In: *Analytical Chemistry* 75 (2003), pp. 4646–4658 (cit. on p. 28).

[106] Alexey I Nesvizhskii. "A survey of computational methods and error rate estimation procedures for peptide and protein identification in shotgun proteomics." In: *Journal of proteomics* 73.11 (Oct. 2010), pp. 2092–123. ISSN: 1876-7737. DOI: 10.1016/j.jprot.2010.08.009. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2956504%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract (cit. on p. 12).

[107] Alexey I Nesvizhskii and Ruedi Aebersold. "Interpretation of shotgun proteomic data: the protein inference problem." In: *Molecular & cellular proteomics : MCP* 4.10 (Oct. 2005), pp. 1419–40. ISSN: 1535-9476. DOI: 10.1074/mcp.R500012-MCP200. URL: http://www.mcponline.org/content/4/10/1419.long (cit. on pp. 21, 23).

[108] Alexey I. Nesvizhskii et al. "A Statistical Model for Identifying Proteins by Tandem Mass Spectrometry". In: *Analytical Chemistry* 75.17 (Sept. 2003), pp. 4646–4658. ISSN: 0003-2700. DOI: 10.1021/ac0341261. URL: http://dx.doi.org/10.1021/ac0341261 (cit. on pp. 1, 81, 104).

[109] Jesper V Olsen et al. "Higher-energy C-trap dissociation for peptide modification analysis". In: *Nature methods* 4.9 (2007), pp. 709–712 (cit. on p. 8).

[110] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 2.0*. Mar. 2002. URL: https://www.openmp.org/wp-content/uploads/cspec20.pdf (cit. on p. 85).

[111] Magnus Palmblad et al. "Automated workflow composition in mass spectrometry-based proteomics". In: *Bioinformatics* 35.4 (Feb. 2019). Ed. by Jonathan Wren, pp. 656–664. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty646. URL: https://academic.oup.com/bioinformatics/article/35/4/656/5060940 (cit. on p. 107).

[112] Judea Pearl. “Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach”. In: *Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, USA, August 18-20, 1982.* Ed. by David L Waltz. AAAI Press, 1982, pp. 133–136. ISBN: 0-262-51051-0. URL: http://www.aaai.org/Library/AAAI/1982/aaai82-032.php (cit. on pp. 33, 84, 168).

[113] Judea. Pearl. *Probabilistic reasoning in intelligent systems : networks of plausible inference*. San Mateo, CA: Morgan Kaufmann Publishers Inc., 1988, pp. 143–237. ISBN: 0-934613-73-7. URL: https://dl.acm.org/citation.cfm?id=52121 (cit. on pp. 29, 83).

[114] Yasset Perez-Riverol. *Toward a sample metadata standard in public proteomics repositories*. Oct. 2020. DOI: 10.1021/acs.jproteome.0c00376. URL: https://pubs.acs.org/doi/abs/10.1021/acs.jproteome.0c00376 (cit. on pp. 111, 132).

[115] Yasset Perez-Riverol et al. “The PRIDE database and related tools and resources in 2019: improving support for quantification data.” In: *Nucleic acids research* 47.D1 (Jan. 2019), pp. D442–D450. ISSN: 1362-4962. DOI: 10.1093/nar/gky1106. URL: https://academic.oup.com/nar/article/47/D1/D442/5160986%20http://www.ncbi.nlm.nih.gov/pubmed/30395289%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6323896 (cit. on p. 88).

[116] Julianus Pfeuffer. *PSM Probabilities from Percolator on merged IDs from four human samples*. Jan. 2020. DOI: 10.5281/zenodo.3531682. URL: https://doi.org/10.5281/zenodo.3531682 (cit. on p. 89).

[117] Julianus Pfeuffer. *UPS2 standard in solution converted to mzML with entrapment and decoy appended protein databases*. Version 1.0. Nov. 2019. DOI: 10.5281/zenodo.3531639. URL: https://doi.org/10.5281/zenodo.3531639 (cit. on p. 89).

[118] Julianus Pfeuffer and Oliver Serang. “A Bounded p-norm Approximation of Max-Convolution for Sub-Quadratic Bayesian Inference on Additive Factors”. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–39. URL: http://jmlr.csail.mit.edu/papers/volume17/15-319/15-319.pdf (cit. on pp. 131, 170).

[119] Julianus Pfeuffer et al. *nf-core/proteomicslfq: v1.0.0 - Lovely Logan [18.10.2020]*. Oct. 2020. DOI: 10.5281/ZENODO.4106005. URL: https://zenodo.org/record/4106005 (cit. on pp. 3, 132).

[120] Lindsay K. Pino et al. “The Skyline ecosystem: Informatics for quantitative mass spectrometry proteomics”. In: *Mass Spectrometry Reviews* 39.3 (May 2020), pp. 229–244. ISSN: 10982787. DOI: 10.1002/mas.21540. URL: https://pubmed.ncbi.nlm.nih.gov/28691345/ (cit. on p. 107).

[121] T. S. Price et al. “EBP, a program for protein identification using multiple tandem mass spectrometry datasets”. In: *Molecular Cell Proteomics* 6.3 (2007), pp. 527–536 (cit. on p. 28).

[122] J. Michael Proffitt et al. “Proteomics in non-human primates: utilizing RNA-Seq data to improve protein identification by mass spectrometry in vervet monkeys”. In: *BMC Genomics* 18.1 (Dec. 2017), p. 877. ISSN: 1471-2164. DOI: 10.1186/s12864-017-4279-0. URL: https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-4279-0 (cit. on p. 104).

[123] Claire Ramus et al. “Spiked proteomic standard dataset for testing label-free quantitative software and statistical methods”. In: *Data in Brief* 6 (Mar. 2016), pp. 286–294. ISSN: 23523409. DOI: 10.1016/j.dib.2015.11.063. URL: https://europepmc.org/articles/PMC4706616%20http://europepmc.org/article/MED/26862574 (cit. on p. 117).

[124] L. Reiter et al. “Protein identification false discovery rates for very large proteomics data sets generated by tandem mass spectrometry”. In: *Molecular and Cellular Proteomics* 8.11 (2009), pp. 2405–2417 (cit. on p. 25).

[125] Lukas Reiter et al. “Protein identification false discovery rates for very large proteomics data sets generated by tandem mass spectrometry”. In: *Molecular & Cellular Proteomics* 8.11 (2009), pp. 2405–2417 (cit. on p. 20).

[126] Gerhard X. Ritter and Joseph N. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, Inc., Aug. 2000. ISBN: 0849300754. URL: http://dl.acm.org/citation.cfm?id=556981 (cit. on p. 42).

[127] Ronald L Rivest, Adi Shamir, and Len Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on p. 32).

[128] V. Rokhlin. “Rapid solution of integral equations of classical potential theory”. In: *Journal of Computational Physics* 60.2 (Sept. 1985), pp. 187–207. ISSN: 10902716. DOI: 10.1016/0021-9991(85)90002-6 (cit. on p. 77).

[129] H.L. Röst et al. “OpenMS: A flexible open-source software platform for mass spectrometry data analysis”. In: *Nature Methods* 13.9 (2016), pp. 741–748. ISSN: 15487105 15487091. DOI: 10.1038/nmeth.3959 (cit. on p. 88).

[130] Hannes L. Röst. *Deep learning adds an extra dimension to peptide fragmentation*. June 2019. DOI: 10.1038/s41592-019-0428-5. URL: https://www.nature.com/articles/s41592-019-0428-5 (cit. on p. 13).

[131] Mikhail M. Savitski et al. “A Scalable Approach for Protein False Discovery Rate Estimation in Large Proteomic Data Sets”. In: *Molecular & Cellular Proteomics* 14.9 (Sept. 2015), pp. 2394–2404. ISSN: 1535-9476. DOI: 10.1074/mcp.M114.046995. URL: http://www.ncbi.nlm.nih.gov/pubmed/25987413%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4563723%20http://www.mcponline.org/lookup/doi/10.1074/mcp.M114.046995 (cit. on p. 104).

[132] James C. Schatzman. “Accuracy of the discrete Fourier transform and the fast Fourier transform”. In: *SIAM Journal of Scientific Computing* 17.5 (Sept. 1996), pp. 1150–1166. ISSN: 10648275. DOI: 10.1137/S1064827593247023. URL: http://epubs.siam.org/doi/10.1137/S1064827593247023 (cit. on p. 46).

[133] Ole Schulz-Trieglaff et al. “LC-MSsim–a simulation software for liquid chromatography mass spectrometry data.” In: *BMC Bioinformatics* 9 (2008), p. 423 (cit. on p. 30).

[134] Michael W. Senko, Steven C. Beu, and Fred W. McLaffertycor. “Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions”. In: *Journal of the American Society for Mass Spectrometry* 6.4 (Apr. 1995), pp. 229–233. ISSN: 18791123. DOI: 10.1016/1044-0305(95)00017-8 (cit. on p. 11).

[135] O. Serang, M. J. MacCoss, and W. S. Noble. “Efficient marginalization to compute protein posterior probabilities from shotgun mass spectrometry data”. In: *Journal of Proteome Research* 9.10 (2010), pp. 5346–5357 (cit. on pp. 30, 81).

[136] Oliver Serang. “A Fast Numerical Method for Max-Convolution and the Application to Efficient Max-Product Inference in Bayesian Networks”. In: *Journal of Computational Biology* (July 2015) (cit. on p. 36).

[137] Oliver Serang. “A Fast Numerical Method for Max-Convolution and the Application to Efficient Max-Product Inference in Bayesian Networks”. In: *Journal of Computational Biology* 22.8 (Aug. 2015), pp. 770–783. ISSN: 10665277. DOI: 10.1089/cmb.2015.0013. URL: https://pubmed.ncbi.nlm.nih.gov/26161499/ (cit. on pp. 41, 42, 45).

[138] Oliver Serang. “Conic Sampling: An Efficient Method for Solving Linear and Quadratic Programming by Randomly Linking Constraints within the Interior”. In: *PLoS ONE* 7.8 (Aug. 2012), e43706 (cit. on pp. 26, 27).

[139] Oliver Serang. “Fast Computation on Semirings Isomorphic to $(\times, \max)$ on $\mathbb{R}+$”. In: *arXiv e-prints* (Nov. 2015). arXiv: 1511.05690. URL: http://arxiv.org/abs/1511.05690 (cit. on pp. 2, 131).

[140] Oliver Serang. “The p-convolution forest: a method for solving graphical models with additive probabilistic equations”. In: *arXiv e-prints* (Aug. 2017), arXiv:1708.06448. arXiv: 1708.06448 [stat.CO] (cit. on p. 91).

[141] Oliver Serang. “The probabilistic convolution tree: efficient exact Bayesian inference for faster LC-MS/MS protein inference.” In: *PloS one* 9.3 (Jan. 2014), e91507. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0091507. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3953406%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract (cit. on pp. 35, 41–43, 72, 84, 85, 131, 167).

[142] Oliver Serang and Lukas Käll. “Solution to Statistical Challenges in Proteomics Is More Statistics, Not Less”. In: *Journal of Proteome Research* 14.10 (2015). ISSN: 15353907. DOI: 10.1021/acs.jproteome.5b00568 (cit. on p. 20).

[143] Oliver Serang, Michael J MacCoss, and William Stafford Noble. “Efficient marginalization to compute protein posterior probabilities from shotgun mass spectrometry data.” In: *Journal of proteome research* 9.10 (Oct. 2010), pp. 5346–57. ISSN: 1535-3907. DOI: 10.1021/pr100594k. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2948606%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract (cit. on p. 42).

[144] Oliver Serang, Michael J MacCoss, and William Stafford Noble. “Efficient marginalization to compute protein posterior probabilities from shotgun mass spectrometry data.” In: *Journal of proteome research* 9.10 (Oct. 2010), pp. 5346–57. ISSN: 1535-3907. DOI: 10.1021/pr100594k. URL: http://dx.doi.org/10.1021/pr100594k (cit. on pp. 83, 84).

[145] Oliver Serang and William Noble. "A review of statistical methods for protein identification using tandem mass spectrometry." In: *Statistics and its interface* 5.1 (Jan. 2012), pp. 3–20. ISSN: 1938-7989. URL: `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3402235%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract` (cit. on p. 21).

[146] Oliver Serang and William Stratford Noble. "Faster Mass Spectrometry-Based Protein Inference: Junction Trees Are More Efficient than Sampling and Marginalization by Enumeration". In: *Computational Biology and Bioinformatics, IEEE/ACM Transactions on* 9.3 (2012), pp. 809–817 (cit. on p. 33).

[147] Oliver Serang et al. "Recognizing uncertainty increases robustness and reproducibility of mass spectrometry-based protein inferences." In: *Journal of proteome research* 11.12 (Dec. 2012), pp. 5586–91. ISSN: 1535-3907. DOI: `10.1021/pr300426s`. URL: `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3534833%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract` (cit. on pp. 23, 105).

[148] C. Shen et al. "A hierarchical statistical model to assess the confidence of peptides and proteins inferred from tandem mass spectrometry". In: *Bioinformatics* 24 (2008), pp. 202–208 (cit. on p. 30).

[149] David Shteynberg et al. "iProphet: multi-level integrative analysis of shotgun proteomic data improves peptide and protein identification rates and error estimates." In: *Molecular & cellular proteomics : MCP* 10.12 (Dec. 2011), p. M111.007690. ISSN: 1535-9484. DOI: `10.1074/mcp.M111.007690`. URL: `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3237071%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract` (cit. on p. 16).

[150] P. Slavík. "A Tight Analysis of the Greedy Algorithm for Set Cover". In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 435–441. ISBN: 0-89791-785-5 (cit. on p. 26).

[151] J. D. Storey. "A direct approach to false discovery rates". In: *Journal of the Royal Statistical Society*. B 64 (2002), pp. 479–498 (cit. on p. 17).

[152] Marc Sturm and Oliver Kohlbacher. "TOPPView: An Open-Source Viewer for Mass Spectrometry Data". In: *Journal of Proteome Research* 8.7 (July 2009), pp. 3760–3763. ISSN: 1535-3893. DOI: `10.1021/pr900171m`. URL: `https://pubs.acs.org/doi/abs/10.1021/pr900171m` (cit. on pp. 10, 14).

[153] Marc Sturm et al. "OpenMS – An open-source software framework for mass spectrometry". In: *BMC Bioinformatics* 9.1 (2008), p. 163. ISSN: 1471-2105. DOI: `10.1186/1471-2105-9-163`. URL: `http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-9-163` (cit. on p. 88).

[154] Ning Sun and Zaifu Yang. *The Max-Convolution Approach to Equilibrium Analysis*. Tech. rep. Institute of Mathematical Economics, Dec. 2002. URL: `http://econpapers.repec.org/RePEc:bie:wpaper:341` (cit. on p. 42).

[155] John EP Syka et al. “Peptide and protein sequence analysis by electron transfer dissociation mass spectrometry”. In: *Proceedings of the National Academy of Sciences of the United States of America* 101.26 (2004), pp. 9528–9533 (cit. on p. 8).

[156] D. L. Tabb et al. “Influence of basic residue content on fragment ion peak intensities in low-energy collision-induced dissociation spectra of peptides”. In: *Analytical Chemistry* 76 (2004), pp. 1243–48 (cit. on p. 13).

[157] H. Tang et al. “A computational approach toward label-free protein quantification using predicted peptide detectability”. In: *Bioinformatics* 22 (2006), e481–e488 (cit. on p. 27).

[158] Stephen Tanner et al. “InsPecT: Identification of Posttranslationally Modified Peptides from Tandem Mass Spectra”. In: *Analytical Chemistry* 77.14 (July 2005), pp. 4626–4639. ISSN: 0003-2700. DOI: 10.1021/ac050102d. URL: http://www.ncbi.nlm.nih.gov/pubmed/16013882%20https://pubs.acs.org/doi/10.1021/ac050102d (cit. on p. 13).

[159] Matthew The and Lukas Käll. “Integrated identification and quantification error probabilities for shotgun proteomics”. In: *Molecular and Cellular Proteomics* 18.3 (Mar. 2019), pp. 561–570. ISSN: 15359484. DOI: 10.1074/mcp.RA118.001018. URL: https://doi.org/10.1074/mcp.RA118.001018 (cit. on p. 115).

[160] Matthew The et al. “A Protein Standard That Emulates Homology for the Characterization of Protein Inference Algorithms”. In: *Journal of Proteome Research* 17.5 (May 2018), pp. 1879–1886. ISSN: 1535-3893. DOI: 10.1021/acs.jproteome.7b00899. URL: http://pubs.acs.org/doi/10.1021/acs.jproteome.7b00899 (cit. on pp. 2, 31, 87, 90, 95).

[161] Matthew The et al. “Fast and Accurate Protein False Discovery Rates on Large-Scale Proteomics Data Sets with Percolator 3.0”. In: *Journal of The American Society for Mass Spectrometry* 27.11 (Nov. 2016), pp. 1719–1727. ISSN: 1044-0305. DOI: 10.1007/s13361-016-1460-7. URL: http://www.ncbi.nlm.nih.gov/pubmed/27572102%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5059416%20http://link.springer.com/10.1007/s13361-016-1460-7 (cit. on p. 104).

[162] Timothy K. Toby, Luca Fornelli, and Neil L. Kelleher. “Progress in Top-Down Proteomics and the Analysis of Proteoforms”. In: *Annual Review of Analytical Chemistry* 9 (June 2016), pp. 499–519. ISSN: 19361335. DOI: 10.1146/annurev-anchem-071015-041550. URL: https://www.annualreviews.org/doi/abs/10.1146/annurev-anchem-071015-041550 (cit. on pp. 6, 21).

[163] Julian Uszkoreit et al. “PIA: An Intuitive Protein Inference Engine with a Web-Based User Interface”. In: *Journal of Proteome Research* 14.7 (July 2015), pp. 2988–2997. ISSN: 1535-3893. DOI: 10.1021/acs.jproteome.5b00121. URL: http://www.ncbi.nlm.nih.gov/pubmed/25938255%20http://pubs.acs.org/doi/abs/10.1021/acs.jproteome.5b00121 (cit. on pp. 1, 21, 131).

[164] Julian Uszkoreit et al. “Protein Inference Using PIA Workflows and PSI Standard File Formats”. In: *Journal of Proteome Research* 18.2 (Feb. 2019), pp. 741–747. ISSN: 1535-3893. DOI: 10.1021/acs.jproteome.8b00723. URL: http://pubs.acs.org/doi/10.1021/acs.jproteome.8b00723 (cit. on p. 92).

[165] Tommi Välikangas, Tomi Suomi, and Laura L. Elo. “A comprehensive evaluation of popular proteomics software workflows for label-free proteome quantification and imputation”. In: *Briefings in Bioinformatics* 19.6 (May 2017), pp. 1344–1355. ISSN: 1467-5463. DOI: 10.1093/bib/bbx054. URL: https://academic.oup.com/bib/article-lookup/doi/10.1093/bib/bbx054 (cit. on p. 109).

[166] Marc Vaudel et al. “A complex standard for protein identification, designed by evolution”. In: *Journal of Proteome Research* 11.10 (2012), pp. 5065–5071 (cit. on p. 19).

[167] B.-J. M. Webb-Robertson et al. “A support vector machine model for the prediction of proteotypic peptides for accurate mass and time proteomics”. In: *Bioinformatics* 24.13 (2008), pp. 1503–9 (cit. on p. 30).

[168] Hendrik Weisser and Jyoti S. Choudhary. “Targeted Feature Detection for Data-Dependent Shotgun Proteomics”. In: *Journal of Proteome Research* 16.8 (Aug. 2017), pp. 2964–2974. ISSN: 1535-3893. DOI: 10.1021/acs.jproteome.7b00248. URL: https://pubs.acs.org/doi/10.1021/acs.jproteome.7b00248 (cit. on p. 115).

[169] J Mitchell Wells and Scott A McLuckey. “Collision-induced dissociation (CID) of peptides and proteins”. In: *Methods in enzymology* 402 (2005), pp. 148–185 (cit. on p. 8).

[170] Mathias Wilhelm et al. “Mass-spectrometry-based draft of the human proteome”. In: *Nature* 509.7502 (2014), pp. 582–587 (cit. on pp. 36, 99).

[171] Mathias Wilhelm et al. “Mass-spectrometry-based draft of the human proteome.” In: *Nature* 509.7502 (2014). ISSN: 1476-4687. DOI: 10.1038/nature13319. arXiv: 209 (cit. on p. 131).

[172] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2862 (2003), pp. 44–60. ISSN: 16113349. DOI: 10.1007/10968987_3. URL: https://link.springer.com/chapter/10.1007/10968987%7B%5C_%7D3 (cit. on p. 126).

[173] Christopher Zach, David Gallup, and Jan-Michael Frahm. “Fast gain-adaptive KLT tracking on the GPU”. English. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, June 2008, pp. 1–7. ISBN: 978-1-4244-2339-2. DOI: 10.1109/CVPRW.2008.4563089. URL: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4563089 (cit. on p. 42).

[174] B. Zhang, M. C. Chambers, and D. L. Tabb. “Proteomic parsimony through bipartite graph analysis improves accuracy and transparency”. In: *Journal of Proteome Research* 6.9 (2007), pp. 3549–3557 (cit. on p. 26).

[175] Bo Zhang et al. “Covariation of Peptide Abundances Accurately Reflects Protein Concentration Differences”. In: *Molecular & Cellular Proteomics* 16.5 (May 2017), pp. 936–948. ISSN: 1535-9476. DOI: 10.1074/mcp.O117.067728. URL: http://www.ncbi.nlm.nih.gov/pubmed/28302922%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5417831%20http://www.mcponline.org/lookup/doi/10.1074/mcp.O117.067728 (cit. on p. 106).

[176] Ning Zhang, Ruedi Aebersold, and Benno Schwikowski. “ProbID: A probabilistic algorithm to identify peptides through sequence database searching using tandem mass spectral data”. In: *Proteomics* 2.10 (Oct. 2002), pp. 1406–1412. ISSN: 16159853. DOI: 10.1002/1615-9861(200210)2:10<1406::AID-PROT1406>3.0.CO;2-9. URL: https://pubmed.ncbi.nlm.nih.gov/12422357/ (cit. on p. 12).

Abbreviations

AUC	area under the curve. 118, 120, 121, R
AWS	Amazon Web Services. 117, 126
BN	Bayesian network. 1, 28
BSA	bovine serum albumine. 117
BSD	Berkeley software distribution. 38
CD	continuous deployment. 116
CI	continuous integration. 116
CLI	command-line interface. 108
CPD	conditional probability distribution. 82
CPT	conditional probability table. 83
DAG	directed acyclic graph. 28
de.NBI	German Network for Bioinformatics Infrastructure. 126
DLL	dynamic link library. 110
DNA	deoxyribonucleic acid. 1, 5
DSL	domain-specific language. 108
EM	expectation-maximization. 29
ESI	electrospray ionization. 7
FDR	false discovery rate. 17–21, 27, 81, 99, 114, Q
FFT	fast Fourier transformation. 35
FTICR	Fourier-transform ion cyclotron resonance. 7
FTP	file transfer protocol. 112
FWHM	full width at half maximum. 115

GUI graphical user interface. 39, 107, 108

HPC high-performance computing. 2, 3, 108, 126, 132

HPLC high-performance liquid chromatography. 7, 9, 11

HSM hierarchical statistical model. 29, 30

ILP integer linear program. 25

IO input/output. 108

KNIME the Konstanz Information Miner. 107, 108

LBP loopy belief propagation. 81

LC liquid chromatography. 7, 11

LP linear program. 26

mRNA messenger RNA. 5

MS mass spectrometry. 1–3, 7, 9, 11, 21

MS/MS tandem mass spectrum. 8–14, 22, Q

OpenMS Open Mass Spectrometry. 2, 38–40, Q

PEP posterior error probability. 20, 111

PP posterior probability. 20

PRIDE PRoteomics IDEntifications database. 111, 112, 126, 132

PSM peptide-spectrum match. 12, 15, 17, 19, 20, 22, 27, 29, 38, 99, 110, 111, 114, 117, 164, 178, 179, 189, 192

QMR quick medical reference. 81

QP quadratic program. 26

RMSD root-mean squared deviation. 118

RNA ribonucleic acid. 1, 5

ROC receiver operating characteristic. 118, 120

RT retention time. 115

SDRF sample-to-data relation format. 111, 112, 124, 125, 132

SDS PAGE sodium dodecyl sulfate polyacrylamide gel electrophoresis. 7

TOPP the OpenMS Pipeline. 39, 107, 110

tRNA transfer RNA. 5

UPS universal protein standard. 117, R

URI uniform resource identifier. 112, 124

A.1 Details on the Projection Method

This appendix establishes a closed-form equation for finding the maximum via the projection method as it is used in the implementation of the algorithm and proves/-conjectures an upper/lower bound of its relative error.

A.1.1 Closed-Form Projection Method for $r = 2$

In general, the computation of both the null space spanning vector $(\gamma_0, \gamma_1, \ldots \gamma_r)$ and of machine-precision approximations for the roots of the polynomial $\gamma_0 + \gamma_1 x + \gamma_2 x^2 + \cdots + \gamma_r x^r$ (which can be approximated by constructing a matrix with that characteristic polynomial and performing eigendecomposition; [57]) are both in $O(r^3)$ for each index m in the result; however, by using a small $r = 2$, we can compute a closed form solution of both the null space spanning vector and of the resulting quadratic roots. This enables faster exploitation of the curve of norms for estimating the maximum value of $u^{(m)}$ (although it doesn't achieve the high accuracy possible with a much larger $r \approx e$). This is equivalent to approximating $\|u^{(m)}\|_{p^*}^{p^*} \approx h_1 \alpha_1^{p^*} + h_2 \alpha_2^{p^*}$, where $h_1 + h_2 = k_m = len(u^{(m)})$.

In this case, the single spanning vector of the null space of

$$\begin{bmatrix} \|u^{(m)}\|_{p^*}^{p^*} & \|u^{(m)}\|_{2p^*}^{2p^*} & \|u^{(m)}\|_{3p^*}^{3p^*} \\ \|u^{(m)}\|_{2p^*}^{2p^*} & \|u^{(m)}\|_{3p^*}^{3p^*} & \|u^{(m)}\|_{4p^*}^{4p^*} \end{bmatrix}$$

will be

$$\begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} \|u^{(m)}\|_{2p^*}^{2p^*} \|u^{(m)}\|_{4p^*}^{4p^*} - \left(\|u^{(m)}\|_{3p^*}^{3p^*}\right)^2 \\ \|u^{(m)}\|_{2p^*}^{2p^*} \|u^{(m)}\|_{3p^*}^{3p^*} - \|u^{(m)}\|_{p^*}^{p^*} \|u^{(m)}\|_{4p^*}^{4p^*} \\ \|u^{(m)}\|_{p^*}^{p^*} \|u^{(m)}\|_{3p^*}^{3p^*} - \left(\|u^{(m)}\|_{2p^*}^{2p^*}\right)^2 \end{bmatrix}$$

and thus $\hat{\alpha} \approx \|u^{(m)}\|_\infty$ can be computed by using the quadratic formula to solve $\gamma_0 + \gamma_1 x + \gamma_2 x^2 = 0$ for x, and computing $\hat{\alpha}$ using the maximum of those zeros: $\hat{\alpha} = {x_{\max}}^{\frac{1}{p^*}}$. When the quadratic is not well defined, then this indicates that the number of unique elements in $u^{(m)}$ is less than 2, and thus cannot be projected uniquely (i.e., $e_m < r$); in this case, the closed-form linear solution can be used rather than a closed-form quadratic solution:

$$\hat{\alpha} = \left(\frac{\|u^{(m)}\|_{4p^*}^{4p^*}}{\|u^{(m)}\|_{3p^*}^{3p^*}} \right)^{\frac{1}{p^*}}.$$

When the closed-form linear solution is not numerically stable (due to division by a value close to zero), then the p^*-norm approximation can likewise be used.

A.1.2 Adapted Piecewise Algorithm Using Interleaved p^* Points

Because the norms must have evenly spaced p^* values in order to use the projection method described above, the exponential sequence of p^* values used in the original piecewise algorithm will not contain four evenly spaced points (which are necessary to solve the quadratic formulation,i.e., $r = 2$). One possible solution would be to take the maximal stable value of p^* for any index (which will be a power of two found using the original piecewise method), and then also computing norms (via the FFT, as before) for $p^* - 3\delta, p^* - 2\delta, p^* - \delta, p^*$; however, this will result in a $4\times$ slowdown in the algorithm, because for every p^*-norm computed via FFT before, now four must be computed. An alternative approach reuses existing values in the 2^i sequence of p^*: for p^* sufficiently large, then the exponential sequence is guaranteed to include these stable p^* values: $\frac{p^*}{4}, \frac{p^*}{2}, p^*$. By considering $\frac{3p^*}{4}$ in p^* candidates, then we can be guaranteed to have four evenly spaced and stable p^* values. This can be achieved easily by noting that

$$\frac{3p^*}{4} = \frac{\frac{p^*}{2} + p^*}{2},$$

meaning that we can insert all possible necessary p^* values for evenly spaced sequences of length four by first computing the exponential sequence of p^* values and then inserting the averages between every pair of adjacent powers of two (and inserting them in a way that maintains the sorted order): $1, 2, 4, 8, 16, \ldots$ becomes $1, 1.5, 2, 3, 4, 6, 8, 12, 16, \ldots$. Thus, if (for some index m) 16 is the highest stable p^* that is a power of two (i.e., the p^* value that would be used by the original piecewise algorithm), then we are guaranteed to use the evenly spaced sequence $4, 8, 12, 16$. By interleaving the powers of two with the averages from the following powers of two, we reduce the number of FFTs to $2\times$ that used by the original piecewise algorithm. For small values of r (such as the $r = 2$ used here), the estimation of the maximum from each sequence of four norms is in $O(4k)$, meaning the total time will still be $k \log(k) \log(\log(k) + 4k \in O(k \log(k) \log(\log(k)))$, which is the same as before. Because the spacing in this formulation is $\frac{p^*}{4}$, and given the maximal root of the quadratic polynomial $\gamma(x_{\max}) = 0$, then $\hat{\alpha} = x_{\max}^{\frac{4}{p^*}}$ (taking the maximal root $x_{\max}$ to the power $\frac{4}{p^*}$ instead of $\frac{1}{p^*}$, which had been the spacing used in the description of the projection method). The null space projection method is shown in **Algorithm 5**.

Algorithm 7 Linear projection of the maximum, using previously computed values $\text{est}_3, \text{est}_4$ for two p^* with a difference of spacing (estimates given in ascending order of their corresponding p's used). The naming of the variables follows the scheme $\text{est}_i = \|u^{(m)}\|_{\frac{i}{4}\text{maxP}}^{\frac{i}{4}\text{maxP}}$. To prevent numeric instabilities, the algorithm checks for division by zero within a tolerance $\tau_{\text{Div}} = 10^{-10}$ (again, a conservative estimate of the machine precision). The return value is a new estimate of the real maximum.

```
1: procedure MAXLIN(est_3, est_4, spacing)
2:     if $|\text{est}_3| > \tau_{\text{Div}}$ then
3:         result ← $\frac{\text{est}_4}{\text{est}_3}$
4:     else
5:         result ← $\text{est}_4$
6:     end if
7:     return $\text{result}^{(1.0/\text{spacing})}$
8: end procedure
```

Algorithm 8 Quadratic projection of the maximum, using previously computed estimates $est_1, est_2, est_3, est_4$ for four equally spaced p in steps of *spacing* (estimates given in ascending order of their corresponding p's used). The naming of the variables follows the scheme $\text{est}_i = \|u^{(m)}\|_{\frac{i}{4}\text{maxP}}^{\frac{i}{4}\text{maxP}}$. To prevent numeric instabilities, the algorithm checks for division by zero within a tolerance $\tau_{\text{Div}} = 10^{-10}$. The return value is a new estimate of the real maximum.

```
1: procedure MAXQUAD(est_1, est_2, est_3, est_4, spacing)
2:     $\gamma_2 \leftarrow \text{est}_1 * \text{est}_3 - \text{est}_2^2$
3:     $\gamma_1 \leftarrow \text{est}_2 * \text{est}_3 - \text{est}_1 * \text{est}_4$
4:     $\gamma_0 \leftarrow \text{est}_2 * \text{est}_4 - \text{est}_3^2$
5:     preRootValue ← $\gamma_1^2 - 4 * \gamma_2 * \gamma_0$
6:     stableQuadratic ← $(\gamma_0 > \tau_{\text{Div}})$ & (preRootValue >= 0.0)
7:     if stableQuadratic then
8:         result ← $(-\gamma_1 + \sqrt{\text{preRootValue}}/(2 * \gamma_2)$
9:     else                                        ▷ Resort to linear projection
10:        result ← maxLin($\text{est}_3, \text{est}_4$)
11:    end if
12:    return $\text{result}^{(1.0/\text{spacing})}$
13: end procedure
```

A.1.3 Accuracy of the $r = 2$ Projection-Based Method

The full closed-form of the quadratic roots used above (which solve the projection when $r = 2$) will be

$$\begin{aligned}
\hat{\alpha} &= \max\left(\left(\frac{-\gamma_1 \pm \sqrt{\gamma_1^2 - 4\gamma_2\gamma_0}}{2\gamma_2}\right)^{\frac{1}{p^*}}\right) \\
&= \max\Bigg(\Bigg(\Bigg(\|u^{(m)}\|_{2p^*}^{2p^*}\|u^{(m)}\|_{3p^*}^{3p^*} - \|u^{(m)}\|_{1p^*}^{1p^*}\|u^{(m)}\|_{4p^*}^{4p^*} \\
&\quad \pm \Bigg((\|u^{(m)}\|_{2p^*}^{2p^*}\|u^{(m)}\|_{3p^*}^{3p^*} - \|u^{(m)}\|_{1p^*}^{1p^*}\|u^{(m)}\|_{4p^*}^{4p^*})^2 \\
&\quad -4\ (\|u^{(m)}\|_{1p^*}^{1p^*}\|u^{(m)}\|_{3p^*}^{3p^*} - \|u^{(m)}\|_{2p^*}^{2p^{*2}})(\|u^{(m)}\|_{2p^*}^{2p^*}\|u^{(m)}\|_{4p^*}^{4p^*} - \|u^{(m)}\|_{3p^*}^{3p^{*2}})\Bigg)^{0.5}\Bigg) \\
&\quad\ \div 2(\|u^{(m)}\|_{2p^*}^{2p^{*2}} - \|u^{(m)}\|_{1p^*}^{1p^*}\|u^{(m)}\|_{3p^*}^{3p^*})\Bigg)^{\frac{1}{p^*}}\Bigg) \\
&= \max\Bigg(\Bigg(\|u^{(m)}\|_{\infty}^{p^*}\Bigg(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*} \\
&\quad \pm \Bigg((\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*})^2 \\
&\quad -4\ (\|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{2p^*}^{2p^{*2}})(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{4p^*}^{4p^*} - \|v^{(m)}\|_{3p^*}^{3p^{*2}})\Bigg)^{0.5}\Bigg) \\
&\quad\ \div 2(\|v^{(m)}\|_{2p^*}^{2p^{*2}} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*})\Bigg)^{\frac{1}{p^*}}\Bigg) \\
&= \|u^{(m)}\|_{\infty}\max\Bigg(\Bigg(\Bigg(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*} \\
&\quad \pm \Bigg((\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*})^2 \\
&\quad -4\ (\|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{2p^*}^{2p^{*2}})(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{4p^*}^{4p^*} - \|v^{(m)}\|_{3p^*}^{3p^{*2}})\Bigg)^{0.5}\Bigg) \\
&\quad\ \div 2(\|v^{(m)}\|_{2p^*}^{2p^{*2}} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*})\Bigg)^{\frac{1}{p^*}}\Bigg)
\end{aligned}$$

where $p^* = \frac{\text{maxP}}{4}$ in the pseudocode (i.e., the maximum numerically stable p^* used by the piecewise algorithm at that index). Note that $\|u^{(m)}\|_{\infty}^{p^*}$ can be factored out because the exponents in every term in the numerator will be $5p^*$ (i.e., $10p^*$ in the square root). Similarly the terms in the denominator each contain $\|u^{(m)}\|_{\infty}^{4p^*}$. Factoring out the maximum value is then the same as operating on scaled vectors v (instead of u) with the maximum entry being 1, and at least one element of value 1.

Furthermore, the denominator $2\gamma_2 \geq 0$; even though the terms summed to compute γ_2 are not exclusively nonnegative, symmetry can be used to demonstrate that every negative term is outweighed by a unique corresponding term:

$$\begin{aligned}
\gamma_2 &= \|v^{(m)}\|_1 \|v^{(m)}\|_3^3 - \left(\|u^{(m)}\|_{2p^*}^{2p^*}\right)^2 \\
&= \left(\sum_i v_i^{(m)}\right)\left(\sum_i v_i^{(m)3}\right) - \left(\sum_i v_i^{(m)2}\right)^2 \\
&= \sum_{i,j} v_i^{(m)} v_j^{(m)3} - \sum_{i,j} v_i^{(m)2} v_j^{(m)2} \\
&= \sum_{i,j} v_i^{(m)} v_j^{(m)2} \left(v_j^{(m)} - v_i^{(m)}\right) \\
&= \sum_i v_i^{(m)} v_i^{(m)2} \left(v_i^{(m)} - v_i^{(m)}\right) + \sum_{i<j} v_i^{(m)} v_j^{(m)2} \left(v_j^{(m)} - v_i^{(m)}\right) + v_j^{(m)} v_i^{(m)2} \left(v_i^{(m)} - v_j^{(m)}\right) \\
&= 0 + \sum_{i<j} v_i^{(m)} v_j^{(m)} \left(v_j^{(m)} - v_i^{(m)}\right)\left(v_j^{(m)} - v_i^{(m)}\right) \\
&= \sum_{i<j} v_i^{(m)} v_j^{(m)} \left(v_j^{(m)} - v_i^{(m)}\right)^2 \\
&\geq 0.
\end{aligned}$$

Thus, for well-defined problems (i.e., when $\gamma_2 \neq 0$), the denominator $2\gamma_2 > 0$, and therefore, the maximum root of the quadratic polynomial will correspond to the term that adds (rather than subtracts) the square root term:

$$\begin{aligned}
\hat{\alpha} = \|u^{(m)}\|_\infty \Bigg(\bigg(& \|v^{(m)}\|_{2p^*}^{2p^*} \|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*} \|v^{(m)}\|_{4p^*}^{4p^*} \\
+ & \Big((\|v^{(m)}\|_{2p^*}^{2p^*} \|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*} \|v^{(m)}\|_{4p^*}^{4p^*})^2 \\
- 4 & \; (\|v^{(m)}\|_{1p^*}^{1p^*} \|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{2p^*}^{2p^* 2})(\|v^{(m)}\|_{2p^*}^{2p^*} \|v^{(m)}\|_{4p^*}^{4p^*} - \|v^{(m)}\|_{3p^*}^{3p^* 2}) \Big)^{0.5} \Bigg) \\
& \div 2(\|v^{(m)}\|_{2p^*}^{2p^* 2} - \|v^{(m)}\|_{1p^*}^{1p^*} \|v^{(m)}\|_{3p^*}^{3p^*}) \bigg)^{\frac{1}{p^*}} \Bigg).
\end{aligned}$$

The relative absolute error is defined as $|\frac{\hat{\alpha}-\|u^{(m)}\|_\infty}{\|u^{(m)}\|_\infty}| = |\frac{\hat{\alpha}}{\|u^{(m)}\|_\infty} - 1|$; therefore, a bound on the relative error of the projection method can be established by bounding

$$\begin{aligned}
s(p^*,k_m) &= \frac{\hat{\alpha}}{\|u^{(m)}\|_\infty}\\
&= \Bigg(\Big(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*}\\
&+ \Big((\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*})^2\\
&-4\ (\|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{2p^*}^{2p^*2})(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{4p^*}^{4p^*} - \|v^{(m)}\|_{3p^*}^{3p^*2})\Big)^{0.5}\Big)\\
&\div 2(\|v^{(m)}\|_{2p^*}^{2p^*2} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*})\Bigg)^{\frac{1}{p^*}}.
\end{aligned}$$

where the length of the $u^{(m)}$ (respectively $v^{(m)}$) is k_m. Using this reformulation, $s = 1$ indicates a zero-error approximation. This can be rewritten to bound its value before taking to the power $\frac{1}{p^*}$:

$$s(p^*,k_m) = t(p^*,k_m)^{\frac{1}{p^*}}$$

where

$$\begin{aligned}
t(p^*,k_m) &= \Big(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*}\\
&+ \Big((\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{4p^*}^{4p^*})^2\\
&-4\ (\|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*} - \|v^{(m)}\|_{2p^*}^{2p^*2})(\|v^{(m)}\|_{2p^*}^{2p^*}\|v^{(m)}\|_{4p^*}^{4p^*} - \|v^{(m)}\|_{3p^*}^{3p^*2})\Big)^{0.5}\Big)\\
&\div 2(\|v^{(m)}\|_{2p^*}^{2p^*2} - \|v^{(m)}\|_{1p^*}^{1p^*}\|v^{(m)}\|_{3p^*}^{3p^*}).
\end{aligned}$$

The extreme values of $t(p^*,k_m)$ can be found by minimizing and maximizing over the possible values of $v^{(m)} \in V = \{v : [0,1]^{k_m} : \exists i, v_i = 1, \exists j, v_j \in (0,1)\}$. The final constraint on v_j in (0,1) is because any v containing only one unique value (which must be 1 in this case since dividing by the maximum element in $u^{(m)}$ to compute $v^{(m)}$ has divided the value at that index by itself ($\exists i, v_i = 1$) will lead to instabilities. When values in v are identical to one another, using $r = 1$ yields an exact solution, and thus solving with $r = 2$ is not well-defined because $\gamma_2 = 0$. Because all elements $v^{(m)p^*} \in [0,1]$ and $p^* \geq 1$, we can perform a change of variables $v_i^{(m)} = v^{(m)p^*}_i$, thereby

k_m	3	4	5	6	7
Minimum	0.935537	0.902161	0.895671	0.880487	0.85343
Maximum	1	1	1	1	1

Table A.1: Exact bounds of $t(k_m)$ for short vectors of length k_m. This table shows the results of numerical minimization techniques performed on the symbolic closed-form of $t(k_m)$ in Mathematica (using `NMinimize`). All $k_m - 1$ entries (excluding the first that was set to 1.0) were left symbolic and constrained to $[0, 1]$, with restriction that the denominator of $t(k_m)$ was nonzero.

eliminating references to p^*:

$$\begin{aligned} t(k_m) \geq \min_{v \in \mathbb{R}^{k_m} : \exists i, v_i = 1, \exists j, v_j \in (0,1)} & \Big(\|v^{(m)}\|_2^2 \|v^{(m)}\|_3^3 - \|v^{(m)}\|_1^1 \|v^{(m)}\|_4^4 \\ & + \Big((\|v^{(m)}\|_2^2 \|v^{(m)}\|_3^3 - \|v^{(m)}\|_1^1 \|v^{(m)}\|_4^4)^2 \\ & - 4\ (\|v^{(m)}\|_1^1 \|v^{(m)}\|_3^3 - \|v^{(m)}\|_2^{2^2})(\|v^{(m)}\|_2^2 \|v^{(m)}\|_4^4 - \|v^{(m)}\|_3^{3^2}) \Big)^{0.5} \Big) \\ & \div 2(\|v^{(m)}\|_2^{2^2} - \|v^{(m)}\|_1^1 \|v^{(m)}\|_3^3). \end{aligned}$$

$$\begin{aligned} t(k_m) \leq \max_{v \in \mathbb{R}^{k_m} : \exists i, v_i = 1, \exists j, v_j \in (0,1)} & \Big(\|v^{(m)}\|_2^2 \|v^{(m)}\|_3^3 - \|v^{(m)}\|_1^1 \|v^{(m)}\|_4^4 \\ & + \Big((\|v^{(m)}\|_2^2 \|v^{(m)}\|_3^3 - \|v^{(m)}\|_1^1 \|v^{(m)}\|_4^4)^2 \\ & - 4\ (\|v^{(m)}\|_1^1 \|v^{(m)}\|_3^3 - \|v^{(m)}\|_2^{2^2})(\|v^{(m)}\|_2^2 \|v^{(m)}\|_4^4 - \|v^{(m)}\|_3^{3^2}) \Big)^{0.5} \Big) \\ & \div 2(\|v^{(m)}\|_2^{2^2} - \|v^{(m)}\|_1^1 \|v^{(m)}\|_3^3). \end{aligned}$$

For small vector lengths, the exact bounds of $t(k_m)$ are shown in **Table A.1**. Notice that the upper bound is fixed, but the lower bound grows monotonically smaller as k_m, the length of the vector considered, increases. For larger vectors, Mathematica does not find optima in a matter of hours, and for arbitrary-length vectors, the Karush-Kuhn-Tucker criteria do not easily yield minima or maxima; however, we do observe that all maxima are achieved by vectors that are permutations (order does not influence the result) of $v = (1, 1, \ldots 1, b, b, \ldots b)$ (again, when only two unique values are found in v, the approximation is exact and thus $\frac{\hat{\alpha}}{\|u^{(m)}\|_\infty} = 1$). Likewise, the minima are achieved by permutations of $v = (1, a, b, b, \ldots b)$. For this reason, we perform further empirical estimation of the bound by randomly sampling vectors of the form $(1, v_2, v_3, \ldots v_{k_m})$ with $k_m - 1$ degrees of freedom (d.o.f.) and sampling vectors of the form $v = (1, a, b, b, \ldots b)$ (with 2 d.o.f.), whose extrema are shown in **Table A.2**.

k_m	4	64	1024
Minimum ($k_m - 1$ d.o.f.)	0.90221268	0.74942834	0.81858283
Maximum ($k_m - 1$ d.o.f.)	0.99999986	0.92482416	0.86795636
Minimum (form $(1, a, b, \dots b)$, 2 d.o.f.)	0.90216688	0.75455478	0.71695386
Maximum (form $(1, a, b, \dots b)$, 2 d.o.f.)	1.00000000	1.0000000	1.00000000

Table A.2: Bounds via random sampling for vectors different in size and type. This table shows the minimal and maximal values resulting from the evaluation of $t(k_m)$ on 10^5 randomly generated vectors (uniform distribution in $[0, 1]$). The first part shows the result for vectors of potentially unconstrained composition, besides one (w.l.o.g. the first) being set to 1.0. The values in the second half were obtained based on vectors of (supposedly) worst-case composition (i.e., of form $(1, a, b, \dots b)$).

At length 64, we see that due to an extreme value scenario, an unconstrained vector scores slightly lower than a vector holding the worst-case pattern $(1, a, b, \dots b)$, because both forms of sampling approach the true lower bound, but one of the unconstrained $k_m - 2$ d.o.f. is slightly closer.

From these results, we conjecture that t is bounded above ≤ 1 (this is achievable at any length k_m by letting v contain exactly two distinct values). In this manner, we achieve our predicted upper bound of 1 regardless of the length k_m. Likewise, we conjecture that at any k_m (not simply the lengths investigated, where this principle is true), the lower bound is given by vectors of the form $(1, a, b, b, \dots b)$. Qualitatively, this conjecture stems from the fact that since the estimate is perfect when v contains exactly two distinct elements, then the worst-case lower bound when v contains three distinct values will concentrate the points at some value far from the other two distinct values. When four distinct values are permitted, then we conjecture that the optimal choice (for minimizing t) of the fourth value will equal the choice for the third distinct value, since that was already determined to be the best point for deceiving the quadratic approximation. From this conjecture, we can then use the fact that the bounds should only grow more extreme as k_m increases, since $\mathbb{R}^1 \subset \mathbb{R}^2 \subset \cdots$ (i.e., lower-dimensional solutions can always be reached in a higher dimension by setting some values to 0). Thus, the minimum for any possible vector should be conservatively bounded below on vectors of the form $(1, a, b, b, \dots b)$ and is achieved by letting k_m approach ∞:

$$\lim_{k_m \to \infty} t(k_m) = \frac{a^4 b - a^3 b^2 - a^2 b^3 + \sqrt{b^2 \left(-a^4 + 3a^3 b - 3a^2 b^2 + a b^3 + (b-1)^3\right)^2} + a b^4 + b^4 - b^3 - b^2 + b}{2b \left(a^3 - 2a^2 b + a b^2 + (b-1)^2\right)}.$$

The minimum value of this expression over all $a \in [0, 1], b \in [0, 1]$ is 0.704 (computed again with Mathematica). Overall, assuming our conjecture regarding the forms of the vectors achieving the minima and maxima, then it follows that $t \in (0.7, 1]$, and the worst-case relative error at the $p^*_{\max}$ contour will be bounded by

$$|t^{\frac{1}{p^*_{\max}}} - 1| < 1 - 0.7^{\frac{4}{p^*_{\max}}}.$$

Appendix B: EPIFANY

B.1 Supplementary methods

B.1.1 Detailed description of the generative Bayesian model

As mentioned in the main manuscript, the parameterization of the probability tables for the binary protein factors R_i ($i = 1 \dots l$ with l being the number of potential proteins in a dataset) is determined solely by the input/grid parameter γ for the protein priors:

$$P(R_i = 1) = \gamma \tag{B.1}$$

$$P(R_i = 0) = 1 - \gamma. \tag{B.2}$$

Evidence in form of e.g., Percolator probabilities p_j (with j ranging from 1 to k, the number of peptides passing filtering) on the peptide-level is incorporated on the peptide factors E_j during initialization by setting $P(E_i = 1) = p_j$. By default, they are multiplied and re-normalized by a low prior (default 0.1) to focus more on high-scoring peptides. Following the assumptions described in the Model section of the main manuscript, the noisy-OR model suggests Equations B.4 and B.5 below for the conditional probability table of a specific peptide ε given the presence of its parent proteins. The binary random variable E_ε denotes the presence of peptide ε while the binary random variables $R_{\varepsilon,1}, \dots, R_{\varepsilon,p}$ represent the presence of the parent proteins of peptide ε. N_ε is a random variable for the total number of present parent proteins of a peptide.

$$N_\varepsilon = \sum_{i=1}^{p} R_{\varepsilon,i} \tag{B.3}$$

$$P(E_\varepsilon = 0 | R_{\varepsilon,1}, \dots, R_{\varepsilon,p}) = P(E_\varepsilon = 0 | N_\varepsilon = n) = (1-\alpha)^n \cdot (1-\beta) \tag{B.4}$$

$$P(E_\varepsilon = 1 | N_\varepsilon = n) = 1 - P(E_\varepsilon = 0 | N_\varepsilon = n) \tag{B.5}$$

Here, n denotes an instantiation of the random variable N_ε and can take values from 0 to p, the total number of parents of the current peptide ε. Note that Equation B.4 is (compared to the original noisy-OR model) additionally exploiting the symmetry arising due to an equal α for all proteins of a peptide.

The optional regularizing priors for the random variable N_ε (the number of proteins that may produce a certain peptide ε) take the following unnormalized form:

$$P(N_\varepsilon = n) = \begin{cases} 1, & \text{for } n = 0 \\ \frac{1}{n}, & n > 0 \end{cases} \tag{B.6}$$

These priors re-weight the conditional probabilities described in Equation B.5 to model a harmonic decay in probabilities based on the number of proteins that might generate a certain peptide. This results in a more uneven distribution of the evidence from peptide ε starting at the most likely producing proteins (based on their beliefs from the rest of the network), especially in conjunction with max-product inference.

B.1.2 Simple message-passing example on a small acyclic protein-peptide graph

For a simple example we assume that for the three proteins $R1$, $R2$ and $R3$, the following peptides $E1$,$E2$,$E3$ were identified with a PSM passing any potential pre-filtering:

- $R1$: $E1$, $E2$
- $R2$: $E2$
- $R3$: $E2$, $E3$

This graph consists of a single connected component to be processed. Although peptide $E2$ is a shared peptide, no indistinguishable group is present in this example. After re-scoring the PSMs with a tool that estimates posterior (error) probabilities and converting them to posterior probabilities if necessary the following values are obtained for the peptides:

- $E1$: 0.9
- $E2$: 0.7
- $E3$: 0.8

Based on the description in the Algorithm section of the main manuscript, a factor graph is built and filled according to the evidence and a certain parameterization with Equations (B.4), (B.5) and (B.6). For simplicity, we assume the following parameterization:

- $\alpha = 0.7$
- $\beta = 0.0$
- $\gamma = 0.5$

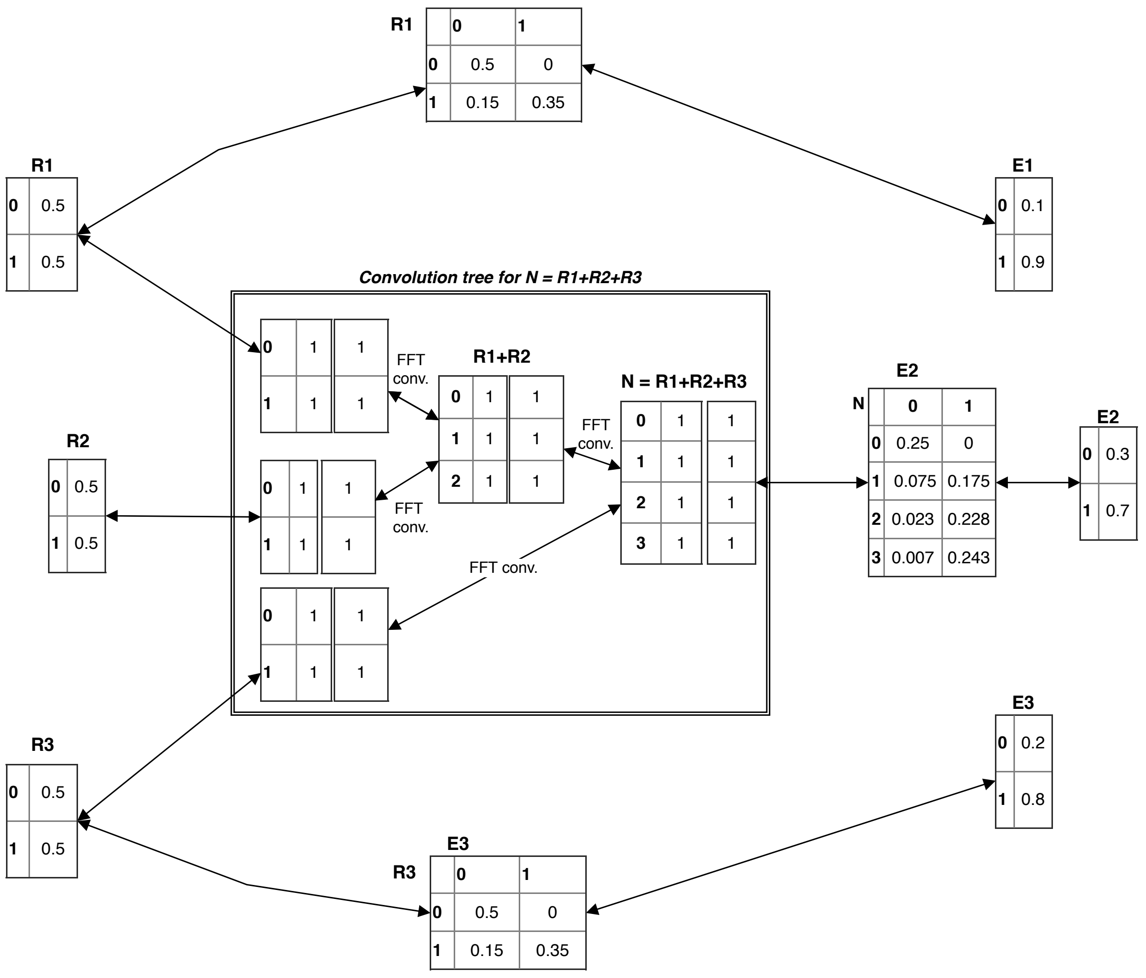


Figure B.1: Factor graph after initialization for the example described in the Supplementary Methods. Random variables of the factors are shown above/besides the corresponding dimensions of the tables holding the initial potentials and later the so-called beliefs. The center inlay for the convolution tree zooms in to an additive factor on the set of random variables $[R1, R2, R3, N]$ to calculate beliefs on the number of parents of peptide $E2$.

- p-norm $= 1$
- Regularization: off
- Damping: off

The resulting graph and initial matrices can be seen in Figure B.1. Under the hood, auxiliary singleton "factors" might be added by the algorithm but are not necessary for the understanding of the general procedure on this example and are therefore omitted. The convolution tree (CT)[141] that mimics an additive factor over $[R1, R2, R3, N]$ is initialized uniformly (shown unnormalized by $\vec{1}$ vectors). Inside the CT, vectors on the left denote beliefs on the number of proteins based on information that is not shared with other participating proteins. Vectors on the right are the likelihoods given the information from shared evidence and the current beliefs of the other participating proteins (shown separated by a double boundary). Due to the graph being acyclic, one pass of messages per direction per edge (plus another one to confirm convergence) suffices to incorporate information across the whole graph. Belief propagation does not need to loop here and will provably result in exact posterior probabilities for the given model[112]. Figures B.2-B.5 will present the steps until posteriors of the proteins have been calculated. The factor beliefs ψ are represented as matrices and have the meaning of their dimensions (each corresponding to one random variable) marked on top. Messages ϕ are passed with information about the intersecting variables between two factors. Messages on top of an edge are passed from left to right and messages below from right to left. The algorithm has to start at factors that carry prior information (e.g. peptides and proteins; the pure R and E factors), depicted in Figure B.2.

Beliefs ψ of a neighboring factor are updated by incoming messages ϕ following the HUGIN algorithm[62] according to the equation below for a message from a factor with random variables (RVs) V to a factor with RVs W whose intersection is S:

$$\phi^{\text{new}}_{V\to W} = p\text{-}\underset{V\setminus S}{\text{marginalize}}\,(\psi_V) \tag{B.7}$$

$$\psi^{\text{new}}_{W} = \frac{\phi^{\text{new}}_{V\to W}}{\phi^{\text{old}}_{V\to W}} \cdot \psi^{\text{old}}_{W} \tag{B.8}$$

p-marginalization is a generalized form of marginalization where instead of summing over the elements from the removed dimensions/variables (equivalent to $p = 1$) the p-norm is computed (denoted $\|\cdot\|_p$). This is done for each set of elements $\mathbf{x}$ that share the same indices in the remaining dimensions:

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \tag{B.9}$$

Note that a $p = \infty$ will result in the maximum over this set and therefore corresponds to max-marginalization as used in max-product message passing algorithms. Currently, for a chosen p all factors perform the same p-norm marginalization to generate a lower-dimensional message out of the higher-dimensional beliefs. Since the first messages (highlighted in red in Figure B.2) pass information only about a single variable that is equal to the one of the factor it originates from, no calculations need to be performed for marginalization and the message equals the prior/evidence. Now incorporation takes

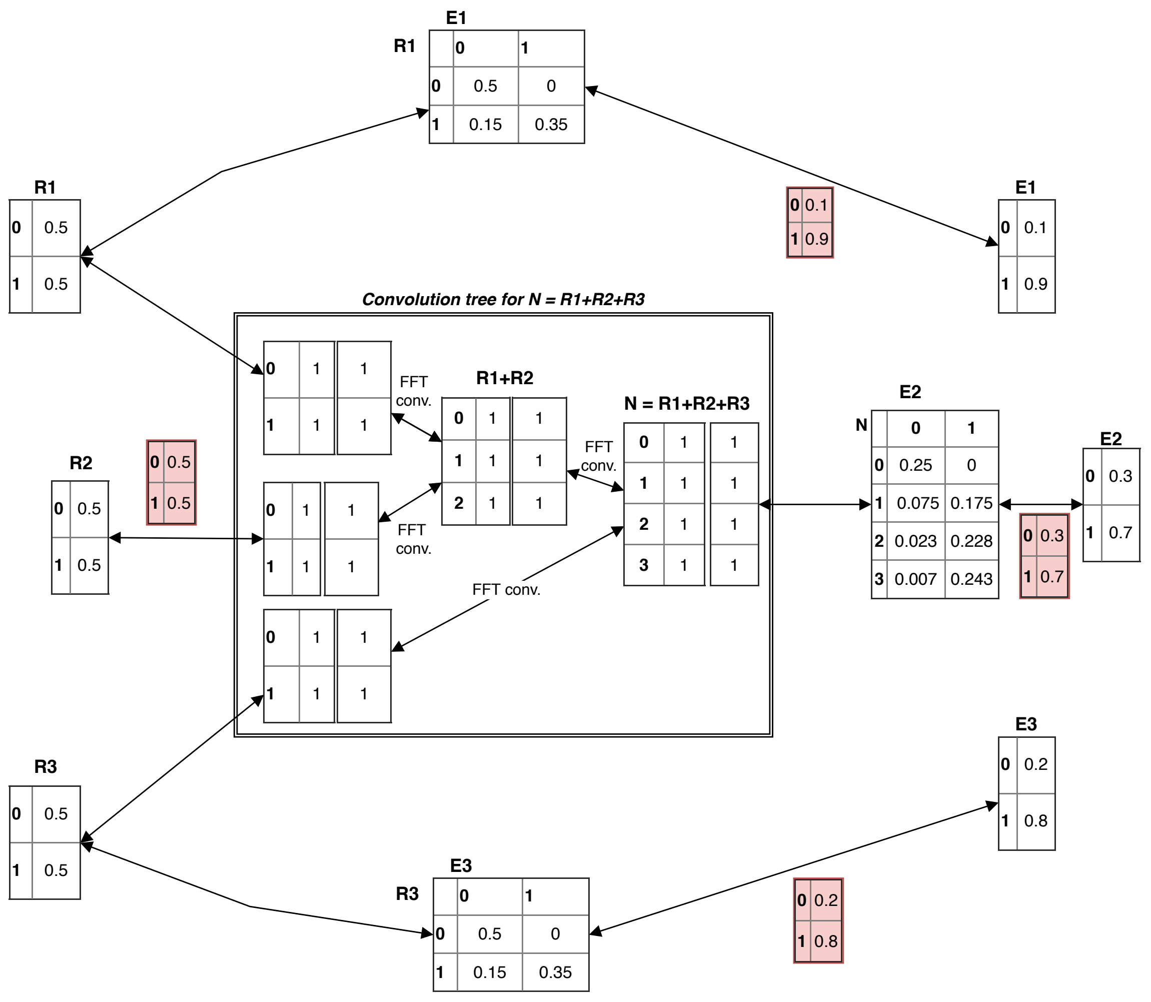


Figure B.2: First (trivial) messages (in red) on the factor graph, according to a manually chosen schedule.

place in the receiving factor by multiplication on the correct dimension (Figure B.3, blue factors) and subsequent re-normalization.

In the same figure, the result of the here used (sum-)marginalization is depicted in form of the outgoing messages (in green) from the recently updated beliefs in $[R1, E1]$, $[R3, E3]$ and $[N, E2]$. As the path to the convolution tree through $R1$ and $R2$ consists of uniformly initialized tables only (protein prior was 0.5), those trivial steps were combined in this example (see additional green tables). The next step (Figure B.4) covers the calculations in the convolution tree which now has enough information to pass outgoing messages (purple tables). Beliefs on the proteins (left vectors) are hierarchically convolved (via fast Fourier transform) to generate beliefs on factor N. Information from the right side is incorporated by convolving the reverse of one “prior” vector (left sides) to generate the “likelihood” (right side) of the other sibling (essentially performing a calculation equivalent to the subtraction of two random variables). The outgoing message (above the edge) on the right side of the CT then consists of the aggregated information from the left side while the messages to each protein on the left side have incorporated information from the right side (the shared peptide $E2$) as well as from every other protein and their unique peptides. Note that p-convolution can be used to model p-marginalization across the CT. Higher values of p result in messages that focus on the information from high-probability configurations (i.e., here the proteins with the highest unique evidence). Even max-product inference ($p = \infty$) is available at low additional computational costs[118]. A higher p is recommended together with the regularized model to have a positive effect on calibration when the degree of ambiguity in the database is suspected to be much higher than in the actual sample.

In the last step (Figure B.5) the messages from the CT need to be incorporated in the protein factors to reach the final protein posteriors, now depicted in gray. Passing this information to the remaining parts of the graph (peptides $E1$ and $E3$) is not shown for brevity.

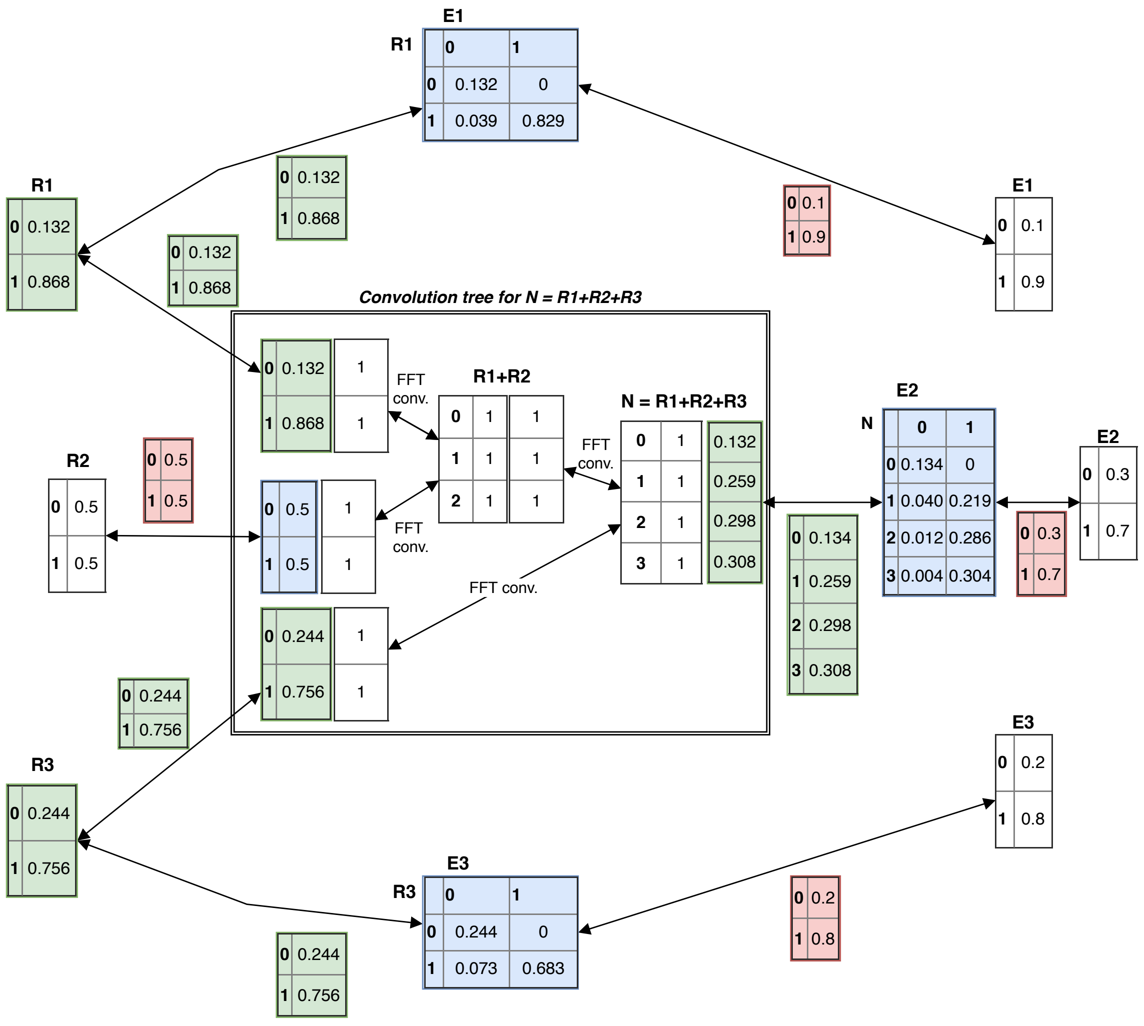


Figure B.3: Incorporation of the first messages through multiplication of probability tables (blue) and subsequent generation of new outgoing messages by (sum-)marginalization (green). It includes the trivial passing of those messages until the convolution tree (also green).

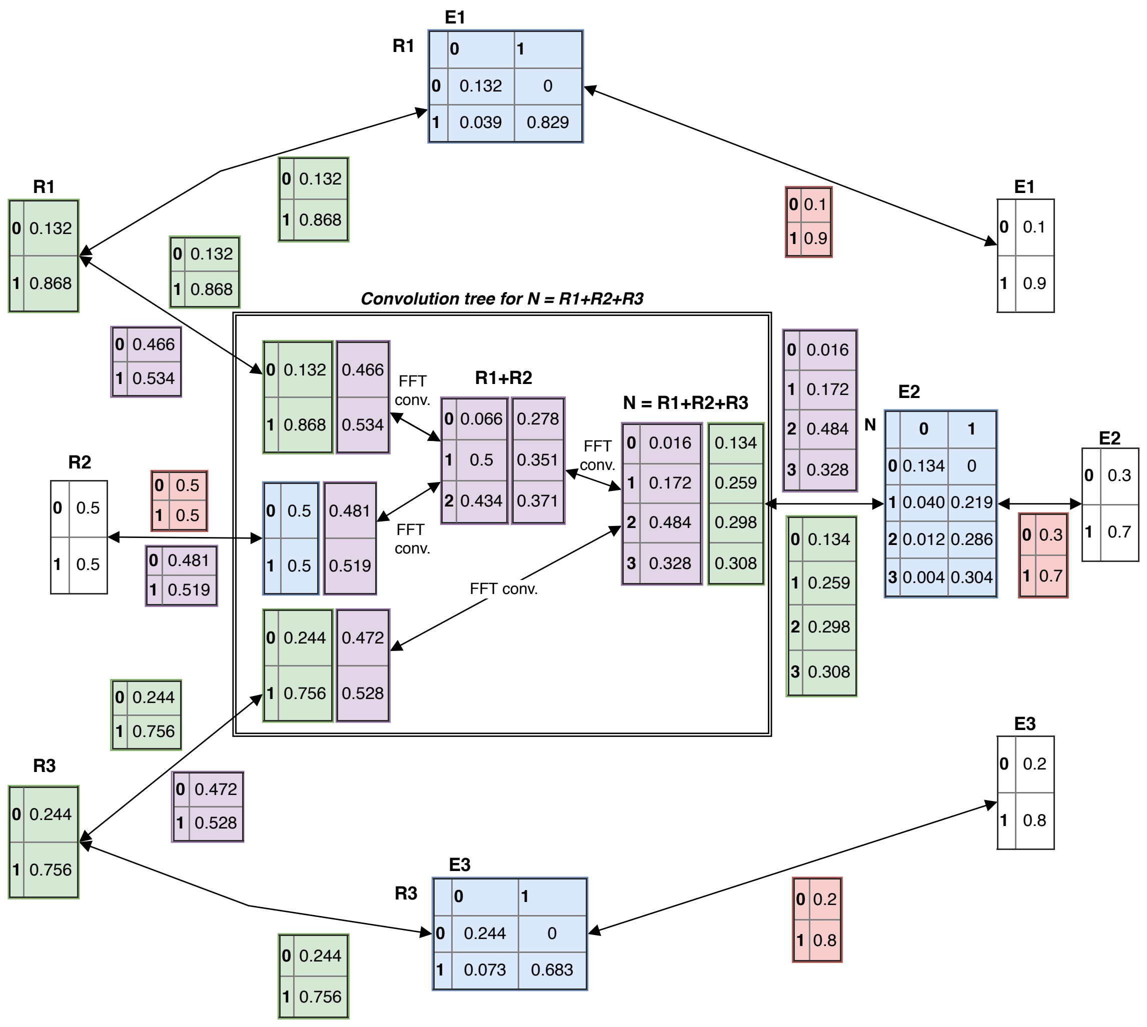


Figure B.4: Calculations on the convolution tree including its outgoing messages (purple). No messages are needed inside. Convolutions (for adding and subtracting random variables) can be performed efficiently by utilizing fast Fourier transforms (“FFT conv.”).

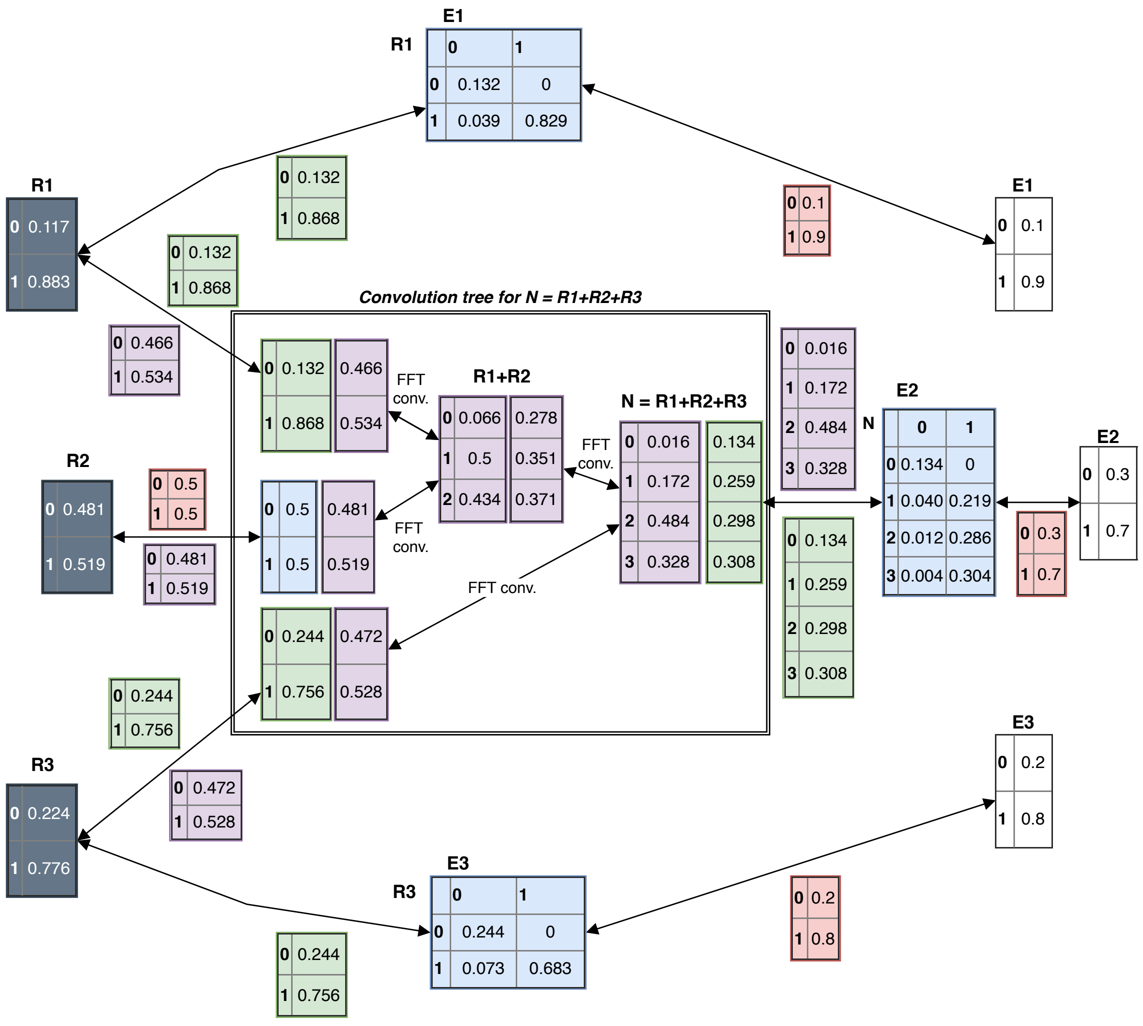


Figure B.5: After incorporation of the messages from the convolution tree (which contained information from $E2$ as well as all other proteins) the beliefs for the proteins will not change anymore since no new information will arrive. $R1$, $R2$ and $R3$ now contain the converged posteriors.

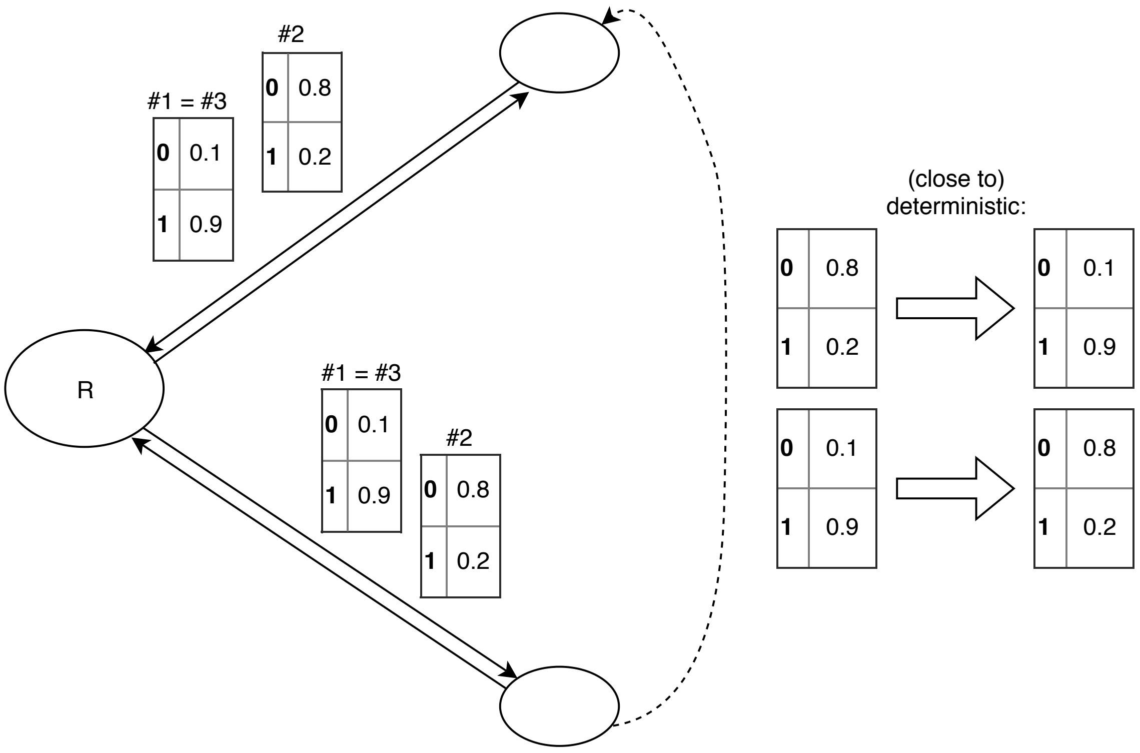


Figure B.6: A cycle closed by a dashed arc (representing a path across potentially many other proteins and peptides) that due to parameterization and observed evidence leads to infinite oscillation of messages to and from the shown protein *R*. The first message (#1) will be the same as the third (#3) and the fifth etc. while the second (#2) will be the same as the fourth and sixth etc. due to the highly deterministic path.

B.1.3 Cycles, damping and convergence

There are cases, where a series of proteins sharing at least one peptide between each pair of them create a cycle such that a protein affects itself according to its current belief. This is often not a problem since after multiple iterations of message passing through the cycle, messages come to an agreement (i.e., the deviation of the values between old and new messages become close to 0). However, there maybe cycles that form messages in such a deterministic way that the beliefs about a protein oscillate indefinitely (Figure B.6).

Assuming that the initial information about a protein results in a message (#1) that holds a high probability for the presence of that protein (0.9) but after the traversal through the cycle it receives a message that states almost the opposite (#2). Then in the second pass through the cycle with the updated information, the old configuration appears again (#3). Intuitively, this may happen in an example like in Figure B.7, where there are three proteins forming a cycle (in a yet untransformed Bayesian network;

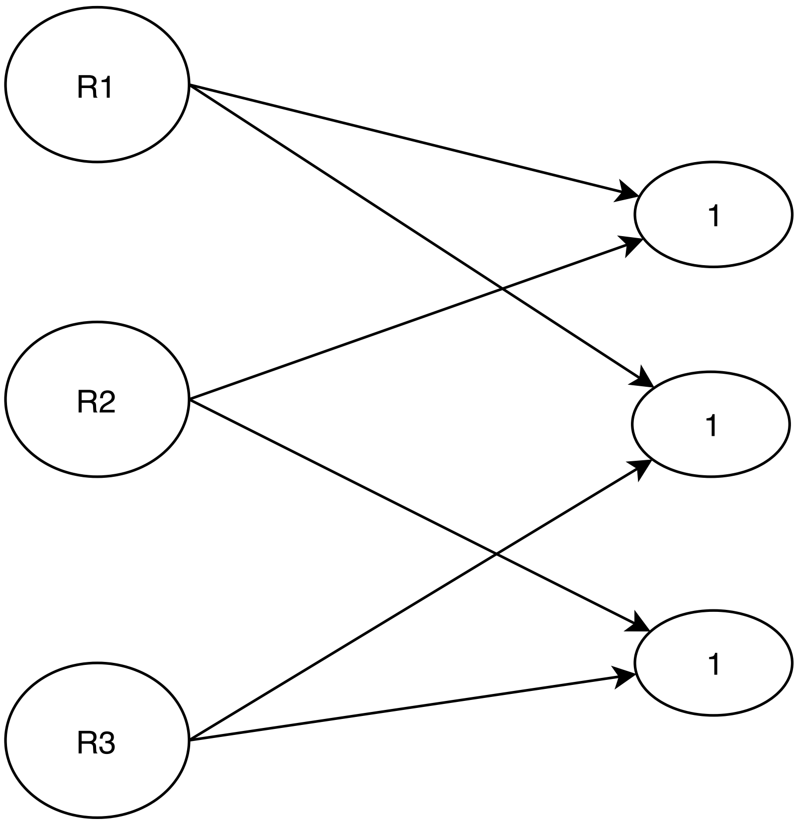


Figure B.7: Three proteins $R1$-$R3$ that with the observed evidence ($P(E_i = 1) = 1, \forall i$ in this component) on the peptide level and the present cycle in the resulting (undirected) factor graph, will lead to oscillations at certain parameterizations.

which is sufficient to show the problem). All peptides were observed with certainty. If the algorithm starts at the lowest peptide, protein $R3$ and $R2$ will receive a high probability for being present. Now since $R3$ has a high probability, $R1$ will be unlikely to be present since its evidence is explained-away. However, a low probability of $R1$ makes $R2$ much more likely which in turn results (at the end of the cycle) in a low probability for $R3$. If the beliefs about $R3$ now enter the cycle again, we will observe the opposite and oscillation will arise.

This disadvantage is due to the partial locality of the algorithm. If it considered all the random variables at the same time, an optimal solution would emerge (but requires exponentially sized tables or at least the calculation thereof). Fortunately, we can perturb the system from time to time and try to force it to another fix-point. That means, to reach convergence faster (or sometimes, as above, to reach it at all), messages can be dampened (by a "momentum" λ) in a way that the updated message is a convex combination of old (if present) and new messages[99]. This is done by point-wise application of the following equation on each element e_i^{new} of message ϕ^{new}

and e_i^{old} of message ϕ^{old}:

$$e_i^{\text{newDamp}} = e_i^{\text{new}} \cdot \lambda + e_i^{\text{old}} \cdot (1 - \lambda) \tag{B.10}$$

including subsequent normalization. Damping can also be performed step-wise in a way reminiscent of simulated annealing, where in the beginning big differences in messages are allowed but later on they are forced to smaller differences by a higher amount of damping. Lastly, whenever a new message would result in a negligible change compared to the last message passed along that edge (i.e., when the point-wise mean-squared error falls under a certain threshold) the algorithm assumes that beliefs converged in this part of the graph and does not queue the new message anymore. Note that messages can be woken up again, if new information arrives from other incoming edges. Global convergence is reached only when the queue of messages is empty.

B.1.4 Parameter optimization

By default, the model parameters α, β, γ are evaluated based on a grid of 42 possible points in the three-dimensional parameter space. Some extreme parameter ranges (close to 0 or 1) were excluded empirically from EPIFANY's defaults due to usually bad classification performance and convergence properties. The rest of the parameter space (except for high values of β which increasingly limit the effect of α) is then covered in multiple steps over each range. The current sets to be combined are:

- γ : 0.2, 0.5, 0.7
- β : 0.01, 0.2, 0.4
- α : 0.1, 0.25, 0.5, 0.65, 0.8

where combinations with $\beta = 0.4$ and $\alpha = 0.1$ are skipped. While the skeleton of the factor graph is kept over multiple sets of parameters, its internals (e.g., the probability tables) are re-initialized for every combination. Based on the converged posteriors of the proteins, the following two measures lead to a decision about the best parameter set:

- pAUC_{100}, the partial target-decoy AUC up to the first 100 decoys.
- $\Delta_{TD,Est}$, the difference between target-decoy FDRs and the FDRs estimated based on the posteriors, measured by the non-overlapping areas of the curves based on trapezoidal integration. The estimated FDR on posteriors for a protein with rank r is defined on the descendingly sorted list of posteriors $(p_1, \ldots, p_n)$ as: $\frac{\sum_{i=1,\ldots,r}(1-p_i)}{r}$.

To create a single objective, the two normalized measures are combined to a convex combination of both: $\lambda \cdot \text{pAUC}_{100} + (1-\lambda) \cdot \Delta_{TD,Est}$. By default, $\lambda = 0.3$.

B.2 Percolator features extracted from search engine results

Tables B.1 and B.2 of the Appendix contain the additionally extracted scores from OpenMS' PSMFeatureExtractor for use in Percolator. PSMFeatureExtractor follows roughly the original extraction scripts (for the different search engines) from the Percolator sources and the implementation was co-developed with one of its authors. Those features were used in addition to basic features like precursor mass deviation and charge.

Score name	Short description
COMET:deltCn	recalculated deltCn = $\frac{(\text{current\_XCorr}-\text{2nd\_best\_XCorr})}{\max(\text{current\_XCorr}, 1)}$
COMET:deltLCn	recalculated deltLCn = $\frac{(\text{current\_XCorr - worst\_XCorr})}{\max(\text{current\_XCorr}, 1)}$
COMET:lnExpect	log(E-value)
MS:1002252	XCorr score
MS:1002255	Sp score = number of candidate peptides
COMET:lnNumSP	log(Sp) = log (number of candidate peptides)
COMET:lnRankSP	log(rank based on Sp score)
COMET:IonFrac	matched ion fraction = matched_ions / total_ions

Table B.1: PSM features extracted from Comet results for use in Percolator. The score name shows the name of the score internally used in OpenMS. Names starting with MS follow the PSI-MS ontology.

Score name	Short description
MS:1002049	reported MSGFPlus RawScore
MS:1002050	reported MSGFPlus DeNovoScore
MSGF:ScoreRatio	RawScore / DeNovoScore
MSGF:Energy	DeNovoScore - RawScore
MSGF:lnEValue	-log(E-Value)
IsotopeError	reported IsotopeError (-1/0/+1)
MSGF:lnExplainedIonCurrentRatio	logarithm of ratio of intensities of matched ions vs. total ion current
MSGF:lnNTermIonCurrentRatio	as above for NTerm ions
MSGF:lnCTermIonCurrentRatio	as above for CTerm ions
MSGF:lnMS2IonCurrent	logarithm of total ion current
MSGF:MeanErrorTop7	mean m/z deviation of the seven most intense fragment ions
MSGF:sqMeanErrorTop7	as above, with squared errors
MSGF:StdevErrorTop7	standard deviation of the above errors

Table B.2: PSM features extracted from MSGFPlus results for use in Percolator. The score name shows the name of the score internally used in OpenMS. Names starting with MS follow the PSI-MS ontology.

B.3 Robustness of the parameter estimation

The robustness of the chose method for parameter estimation can be inferred by the comparable performance on repeated runs of EPIFANY on the iPRG2016 sample B with differently shuffled decoy databases (Figure B.8).

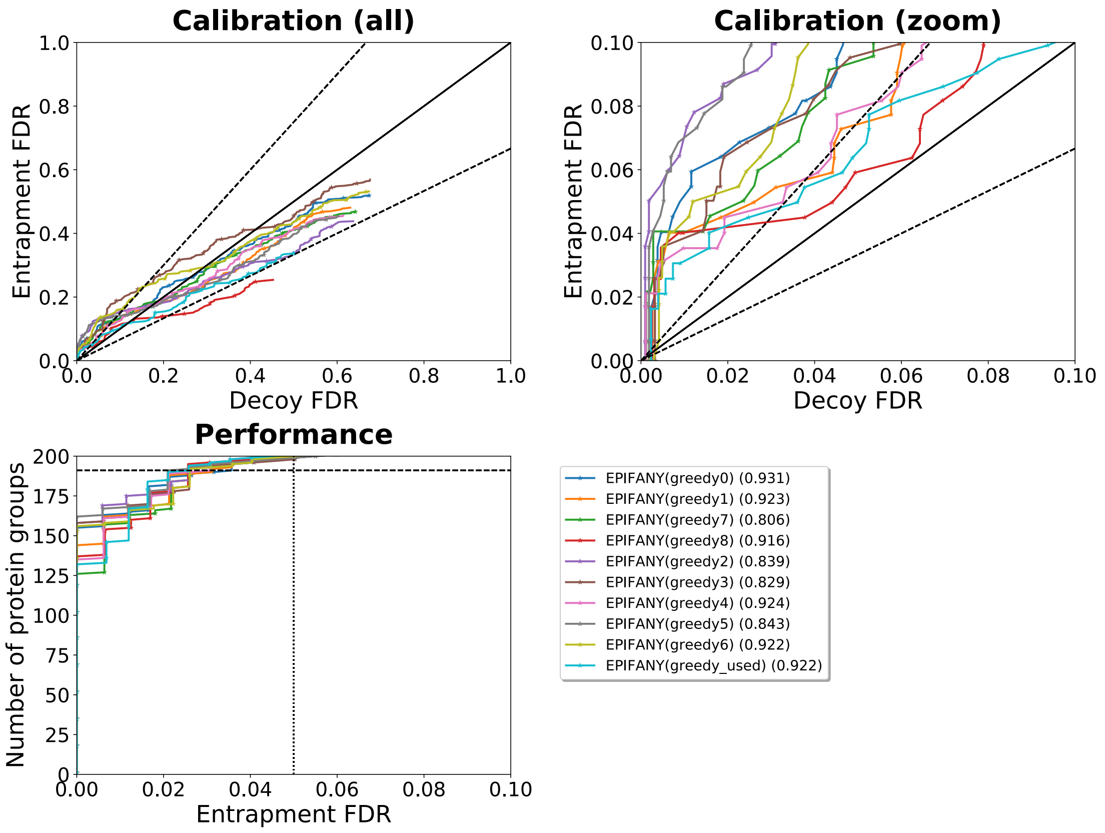


Figure B.8: Different runs of EPIFANY (with greedy resolution) on the iPRG2016 sample B with differently shuffled decoy databases (i.e., different random seeds).

B.4 Supplementary Results

B.4.1 Detailed evaluation plots on iPRG2016

This subsection shows the results on the merged replicates of each sample from the official iPRG2016 evaluation script (adapted to show multiple methods). The log-log calibration subfigure was removed in favor of a legend. In the corresponding Figures B.9-B.19 “RegMaxProd” means a regularized model with max product inference was used. “Greedy” in the label shows that greedy group resolution based on posteriors was enabled. Methods with “OldFDR” use a different equation to estimate FDRs based on target-decoy estimation [D/(T+D) instead of the more conservative (D+1)/T]. Partial AUCs up to a 5% protein target-decoy FDR are noted in parentheses next to the entries in the legend. Note that the Performance subplot shows a somewhat version of the ROC curve since the x-axis represents the (entrapment) FDR. Sample B was the initial choice for evaluation, therefore it contains more tested combinations than follow-up analyses on the other samples. For example, the more conservative FDR approach of EPIFANY was chosen for its better calibration at low FDRs. ProteinProphet was in some plots discarded due to its bad performance hinting at incompatibilities with our pipeline and/or problems during conversion. Initially, several configurations for PIA were tested as well (direct Percolator PEPs, Percolator PEPs with PIA's PSM FDR, Percolator PEPs with Percolator's PSM FDR, Percolator PEPs with OpenMS' FDR, PIA's "report all" option). However, both with OpenMS' FDR and with the "report all" option it completely underperformed and the remaining configuration performed very similar. We received many infinite values when running on OpenMS' FDR as input (maybe due to possible values of 0) and we think the missing parsimony approach in "report all" is detrimental to performance as well and not recommended by the authors anyway. Therefore, we only tested "PIA" with its own PSM FDR and "PIA_PEP" directly on Percolator's PEPs together with the other methods. Due to passing PEPs directly being the fairest comparison, this was chosen for the Figures in the main manuscript. Concerning grouping, since the evaluation script allows groups and counts them favorably when mixed true positive and false positive groups occur, it is only fair to run all methods with grouping (also due to the fact that most methods group by default). This means for Fido, EPIFANY and ProteinProphet, grouped results are the default, if not suffixed otherwise in the figures. All samples were evaluated with the same concatenated target-decoy database as described in the main manuscript (including all A,B and random entrapment proteins plus background). Except for sample B with extended Comet search space, all database searches use the same settings as well. In general, the following properties of the different methods can be described:

- Fido: It shows good performance on most levels for both A/B and A+B but lacks calibration in A/B. However, it seems to sometimes struggle with strictly filtered data.

- PIA: Both good performance and calibration on A/B (more so on B) and A+B but struggles with noise (e.g. through extended search spaces or unfiltered PSMs). Also, it does not identify as many proteins as non-greedy Bayesian methods on A+B. Its "report all" mode fails almost completely on B but performs at least as well as non-greedy methods on A+B. There is often little difference between PIA used directly on PEPs (PIA_PEP) and PIA including its own FDR.

- Percolator: Always well-calibrated and robust to filtering but is very conservative in the amount of proteins it identifies (especially on A+B). It sometimes also underperforms (e.g. on B and B extended) more pronounced.

- EPIFANY: Very robust on all datasets and levels. With its greedy resolution activated it shares many features with PIA except that is more robust to extended search spaces and loose filtering, although it is sometimes slightly outperformed by PIA in the low FDR ranges on highly filtered PSMs as well.
  Without greedy resolution and regularization it behaves very much like Fido due to the then equal models. However, it is more robust at strict cutoffs and performance therefore never drops significantly on the datasets.
  EPIFANY also offers regularization with max-product inference without greedy resolution (labeled “regMaxProd” in the figures) which acts as a trade-off between calibration on datasets where a lot of false proteins can potentially be identified by shared peptides and performance on datasets where most of the proteins are actually present. However—as evident in the results on this dataset—on extreme datasets where either all or only one of the parents are present (A+B and A/B respectively) it achieves the goals not yet fully (i.e. loses calibration at intermediate FDRs on A/B and recovers the last proteins in A+B only at intermediate FDRs as well.)

Sample B

In sample B of the iPRG2016 dataset only proteins from B are known to be present but also proteins from A which share many peptides with B by design were considered during database search. Greedy (EPIFANY with greedy resolution), parsimonious (PIA) and “unique peptide” (Percolator) methods show the best calibration here. Nonetheless, also our regularized version without a greedy resolution ranks most of the true positives higher than most entrapment proteins such that they can be identified at moderate entrapment FDRs while the output target-decoy FDRs are still as calibrated as greedy methods up to a 2% entrapment FDR. While Fido struggles on strict cutoffs, PIA does so on unfiltered data. We also can not recommend PIA’s “report all” mode on this sample (only shown for a cutoff at 5% FDR).

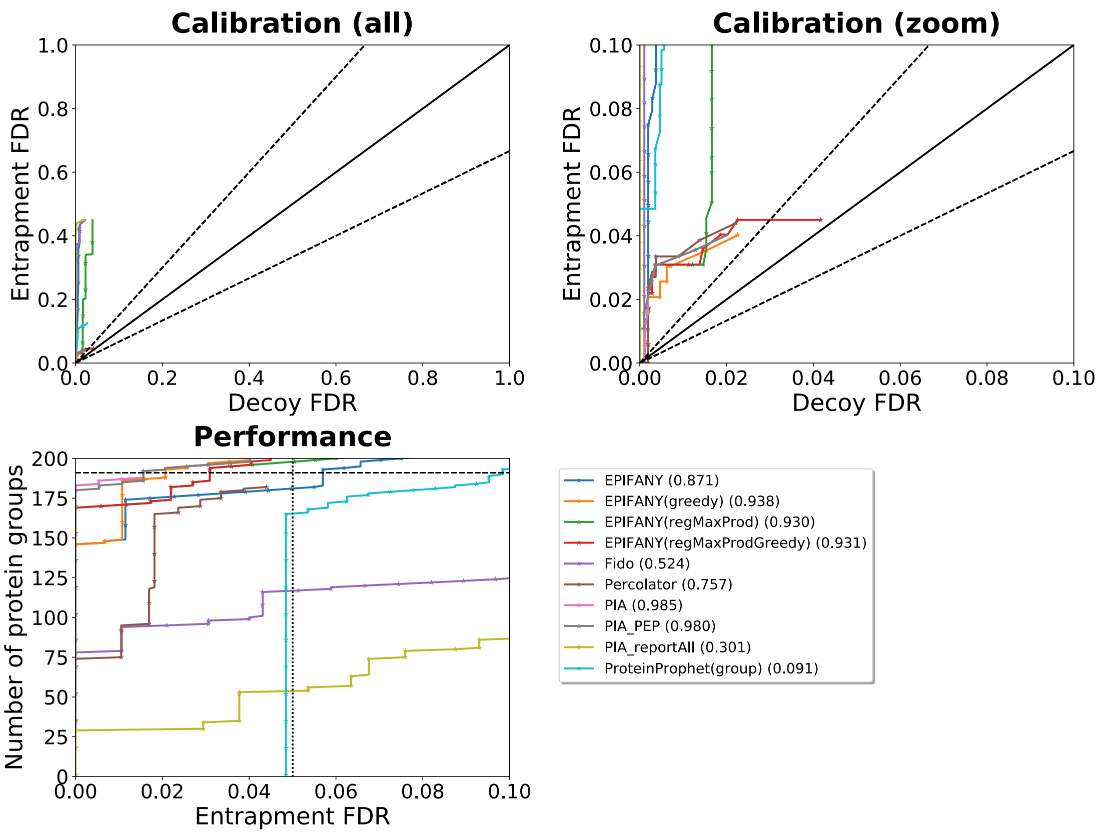


Figure B.9: Results on iPRG2016 sample B (PSM cutoff at 0.01 FDR)

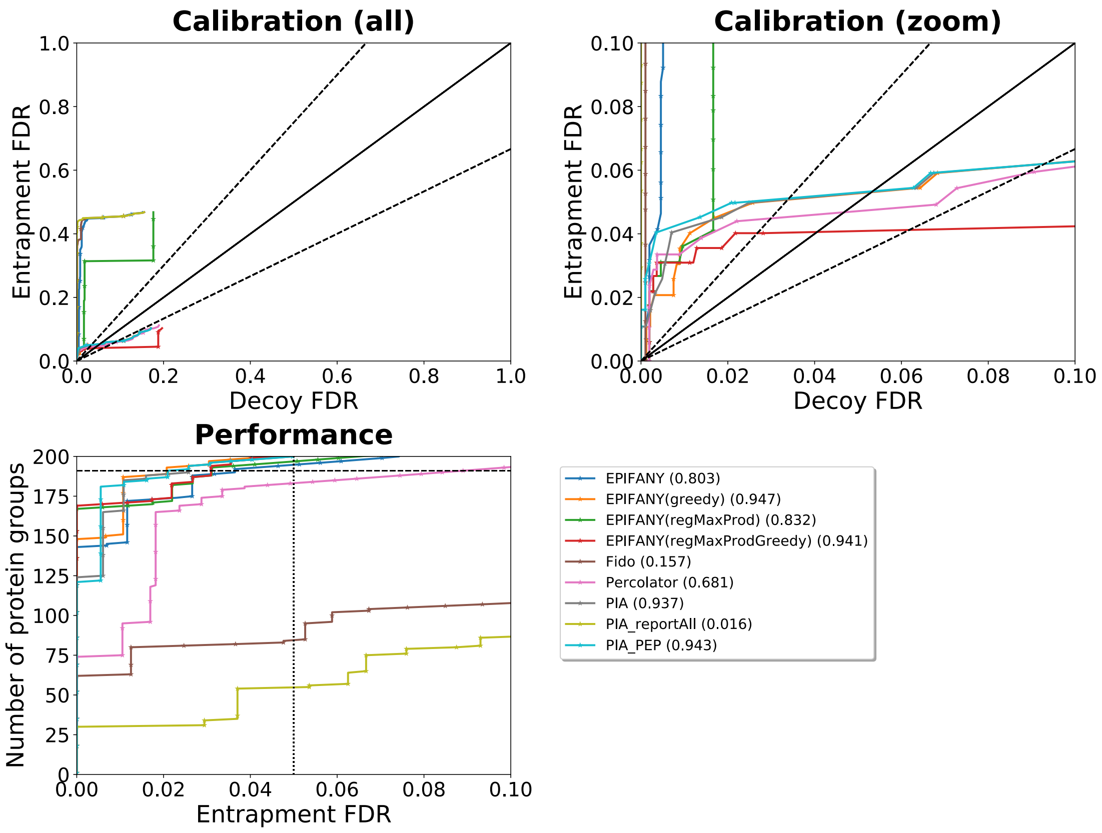


Figure B.10: Results on iPRG2016 sample B (PSM cutoff at 0.05 FDR)

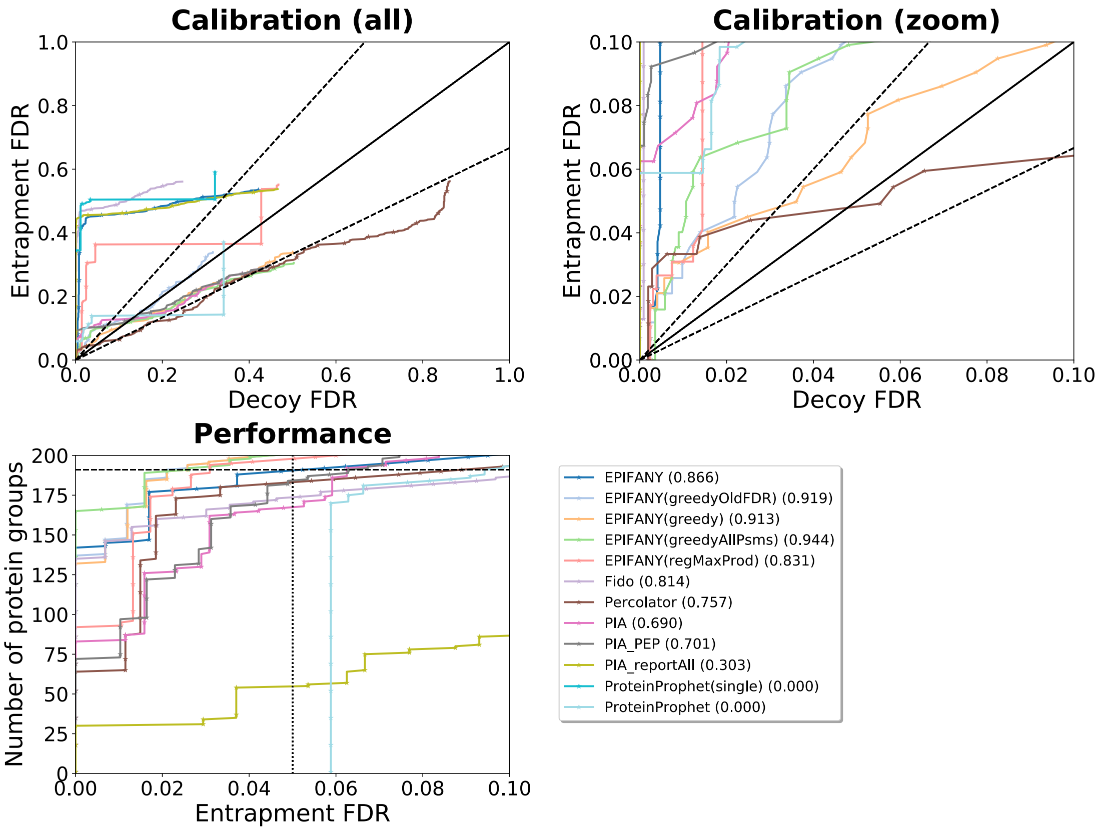


Figure B.11: Results on iPRG2016 sample B (PSM cutoff at 0.001 posterior probability)

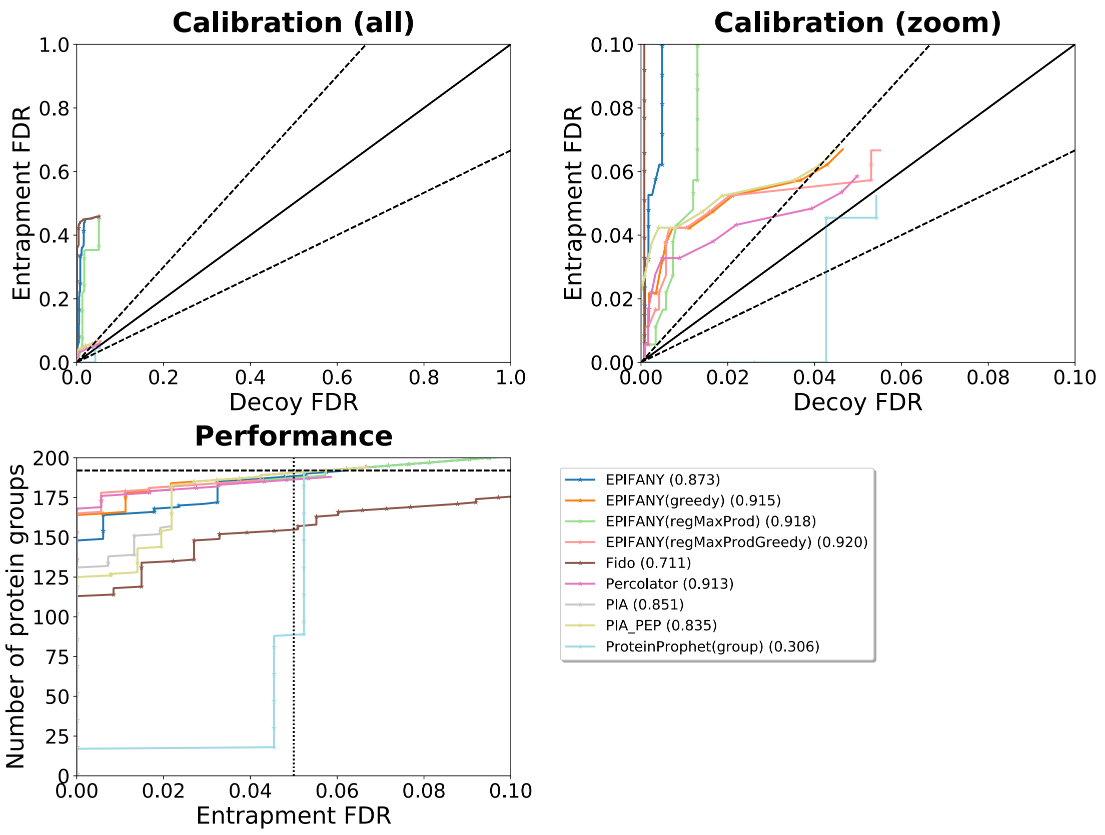


Figure B.12: Results on iPRG2016 sample A (PSM cutoff at 0.01 FDR)

Sample A

Results on sample A roughly confirm the results from sample B except that PIA performs slightly worse on all levels at low FDRs.

Sample AB

When for many shared peptides, multiple or all parent proteins are present, results are different. All approaches yield a rather conservative estimate of the FDR when compared to the entrapment FDR and except for Fido and (non-greedy) EPIFANY the methods are usually not able to identify all true positives at reasonable entrapment FDRs. Non-greedy regularized EPIFANY provides a trade-off here and identifies additional true proteins at intermediate FDRs. PIA's "report all" mode works also well on this sample with the used evaluation metric. However, this sample is an extreme case, where overcounting of shared evidence is beneficial and pushes proteins from the true part of the entrapment database to the top of the results.

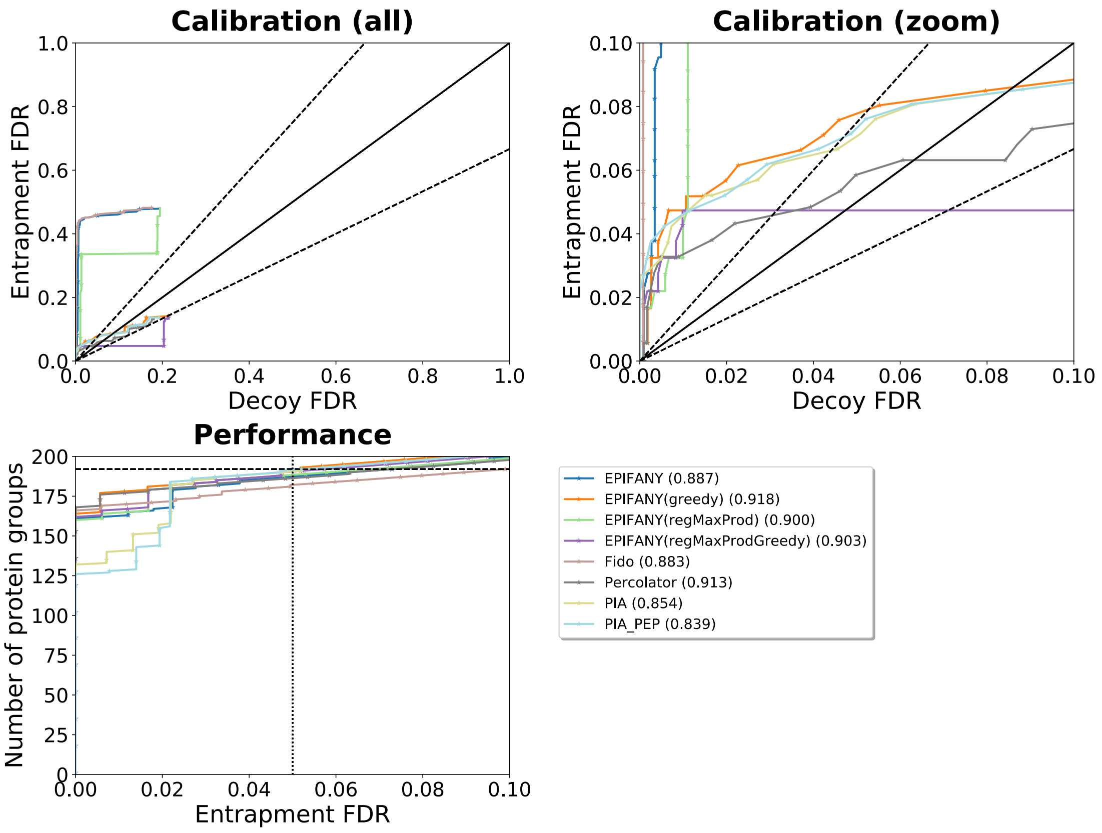


Figure B.13: Results on iPRG2016 sample A (PSM cutoff at 0.05 FDR)

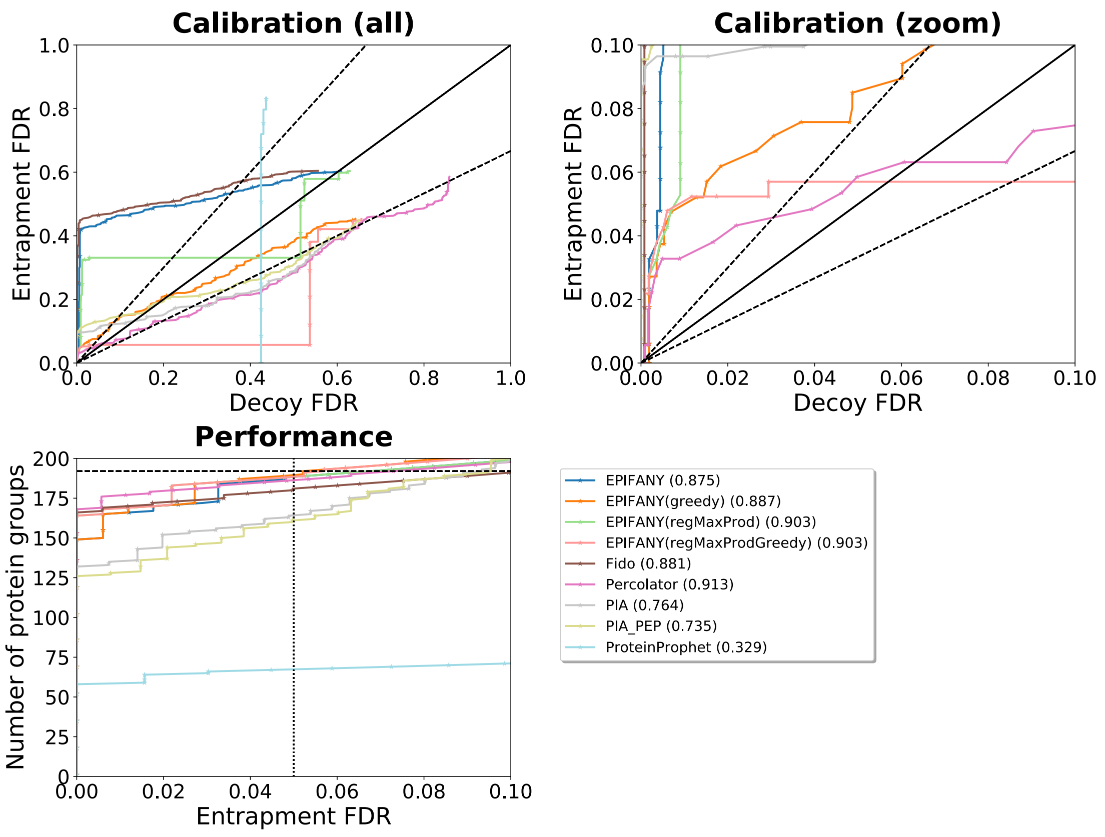


Figure B.14: Results on iPRG2016 sample A (PSM cutoff at 0.001 posterior probability)

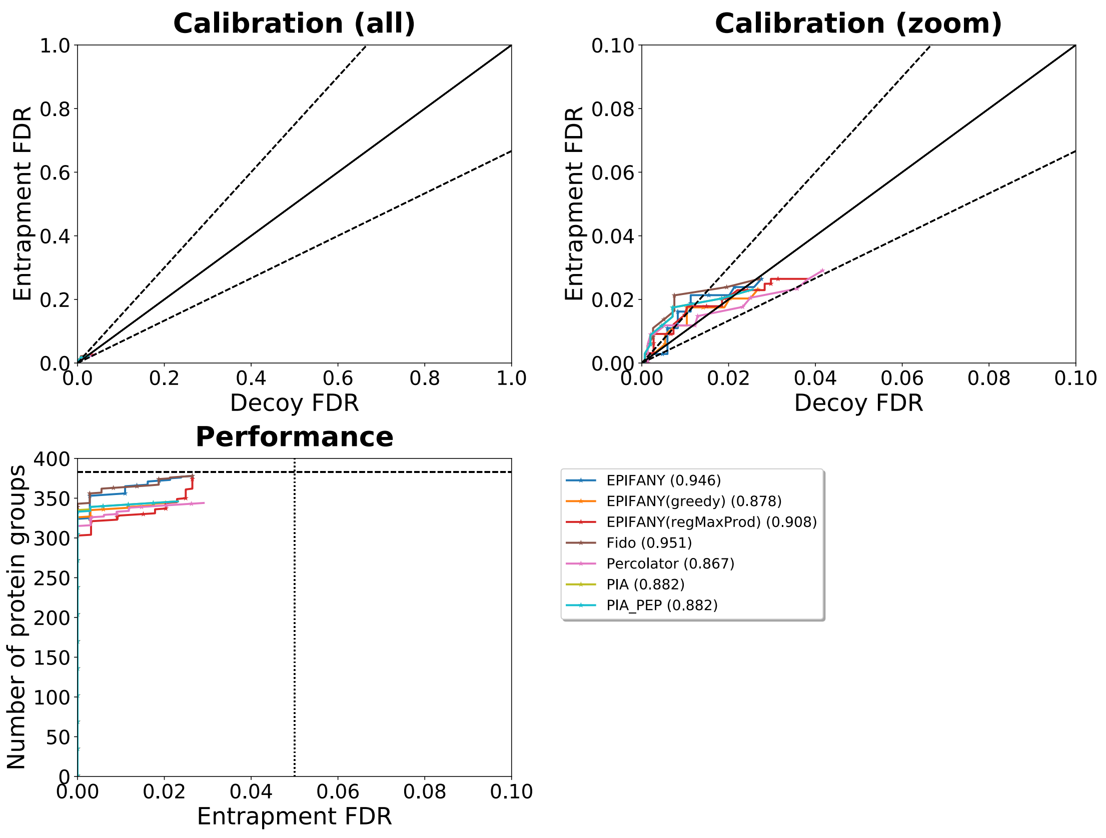


Figure B.15: Results on iPRG2016 sample AB (PSM cutoff at 0.01 FDR)

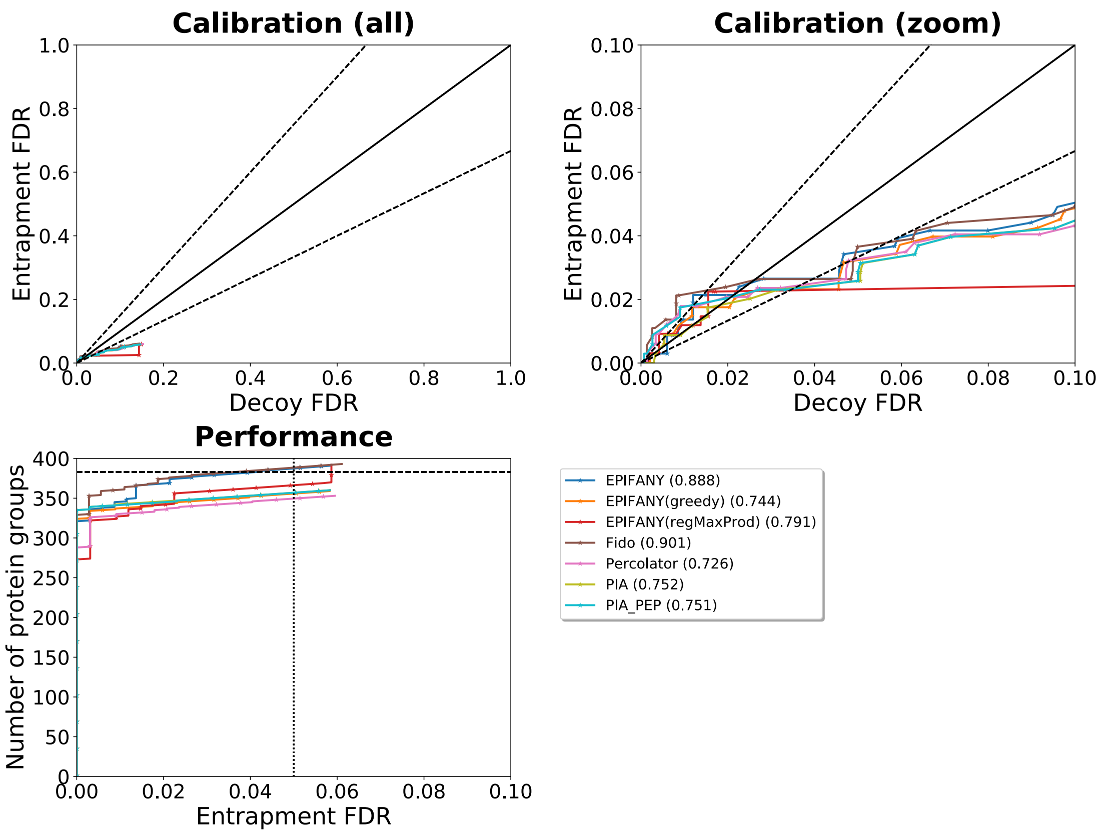


Figure B.16: Results on iPRG2016 sample AB (PSM cutoff at 0.05 FDR)

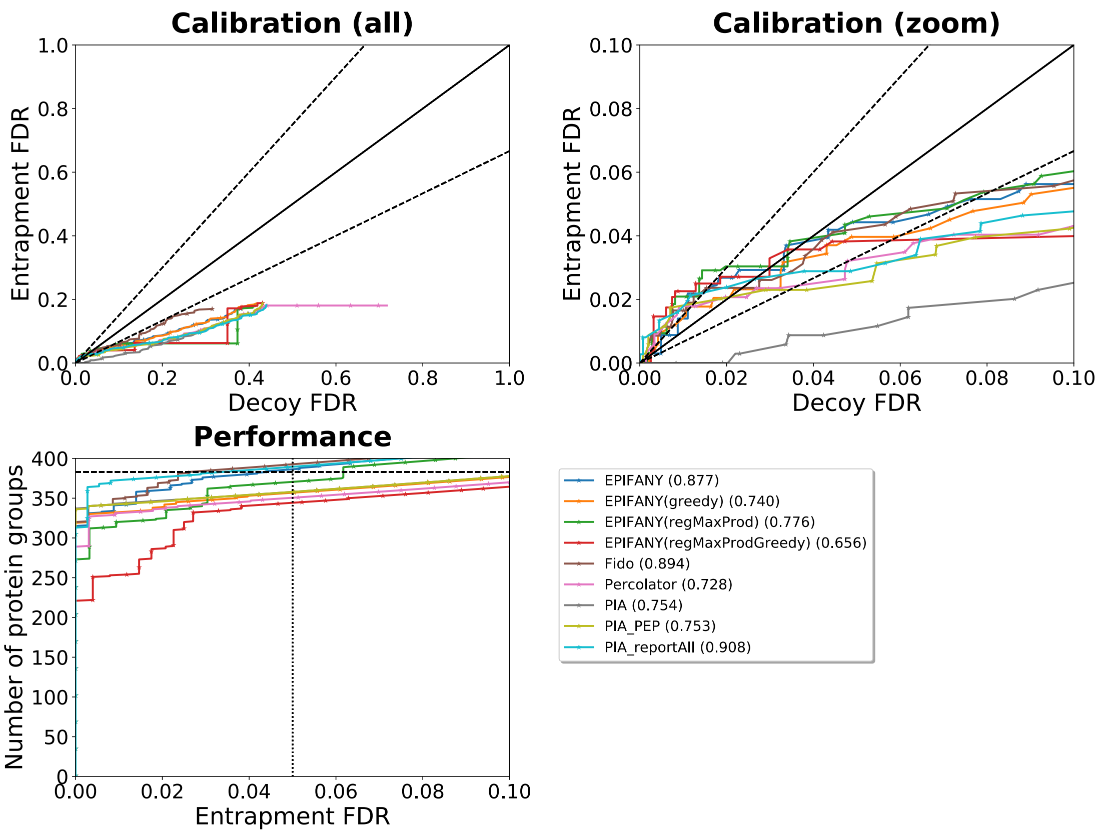


Figure B.17: Results on iPRG2016 sample AB (PSM cutoff at 0.001 posterior probability)

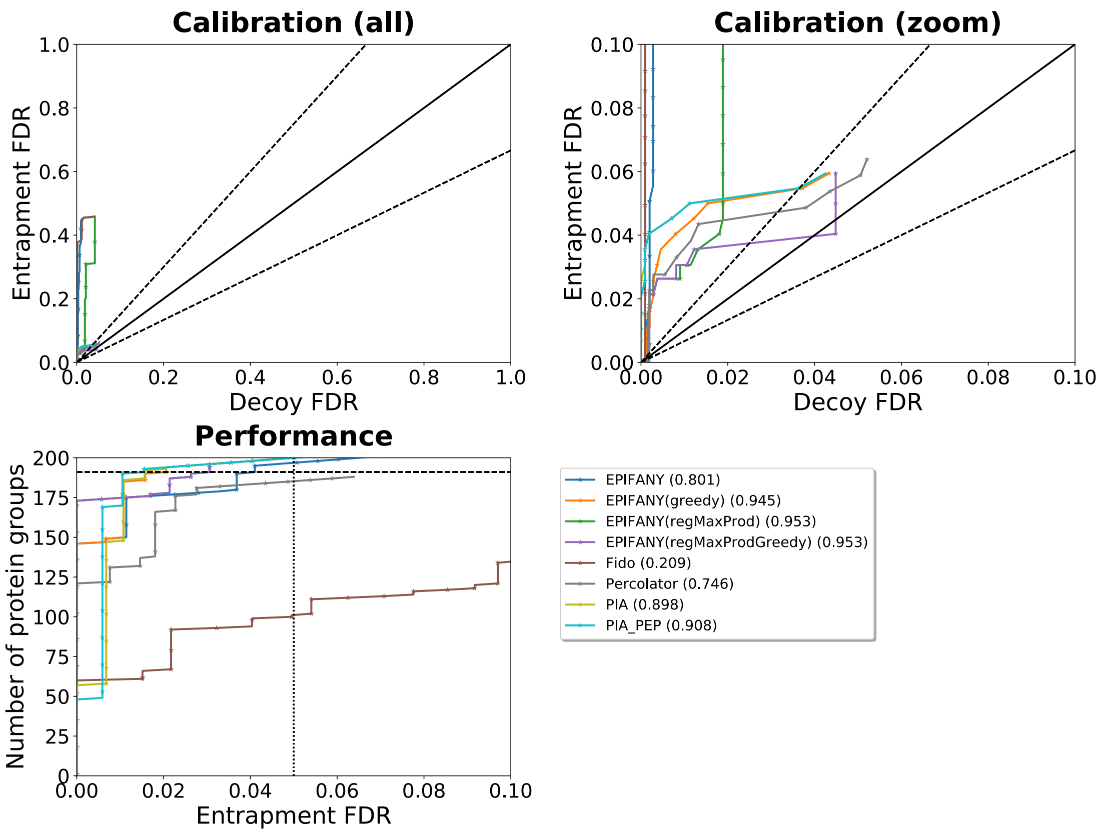


Figure B.18: Results on iPRG2016 sample B with an extended search space set in Comet (PSM cutoff at 0.01 FDR)

Sample B with extended Comet search space

When opening up the search space for Comet's database search (by allowing 50 ppm precursor tolerances, 4 missed cleavages and phosphorylations on S,T,Y), PIA starts to include some false entrapment proteins in the beginning of the ranking while other methods seem to gain slightly in performance.

B.4.2 Robustness of parameter estimation for different decoy databases

The robustness was tested again on the results on the "B" sample from the iPRG2016 dataset (Appendix Figure B.8). Combined PSMs of the three replicates were filtered to include only PSMs with ≥ 0.001 Percolator probability. Different runs of the Comet, Percolator, EPIFANY pipeline were carried out with differently seeded, randomly shuffled decoy databases to show the robustness of the parameter estimation based on protein target-decoy FDRs. Different best parameters were estimated in the repeated runs but spanned a narrow subspace of the considered grid. Larger difference in the calibration can be explained by the relatively large impacts of the ranks of few false positives on

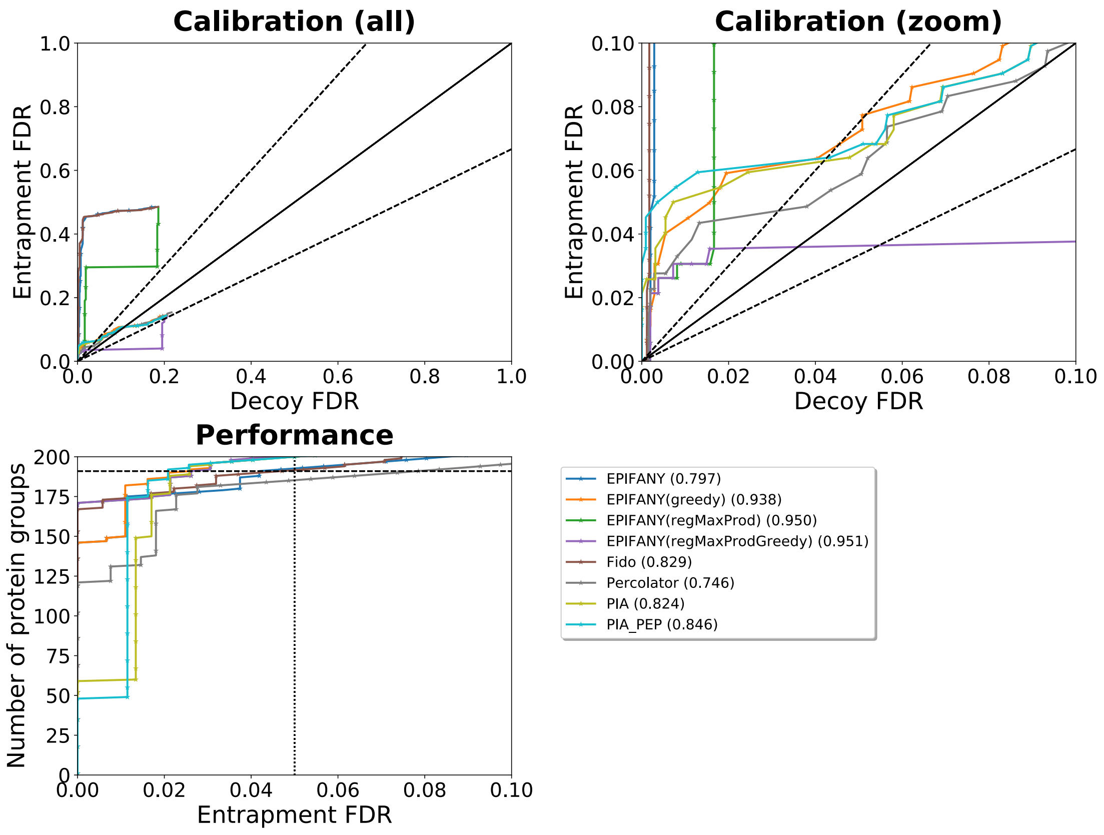


Figure B.19: Results on iPRG2016 sample B with an extended search space set in Comet (PSM cutoff at 0.05 FDR)

such a small set of true positives. The randomly chosen run labeled “greedy_used” used the same decoy database as in Figures B.9-B.19 and is evaluated together with the other methods in the main manuscript.

B.4.3 Additional summary results on iPRG2016 A/B at 1% protein entrapment FDR

Appendix Figure B.20 shows the same results as in the main text for the stricter cutoff of 1% protein group FDR.

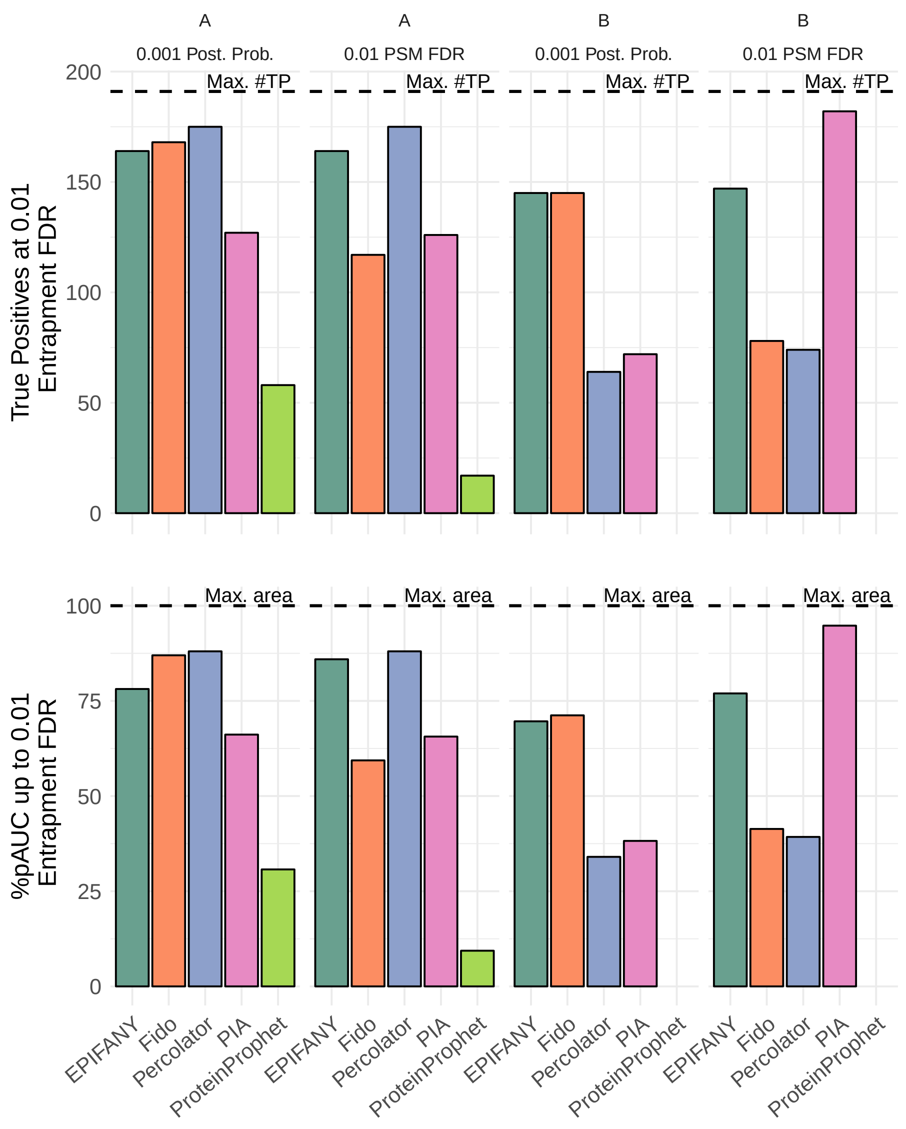


Figure B.20: Summary statistics on the identification performance of different inference methods at a 1% protein entrapment FDR on the iPRG2016 dataset, samples “A” and “B”. PSMs were filtered at two distinct levels (at 0.001 posterior probability and a stricter 0.01 PSM FDR). **Upper:** Number of true positive proteins found. The maximum number of true positives according to the ground truth database given is 191 for “B” and 192 for “A” as indicated by a dashed horizontal line. **Lower:** The percentage of the maximum area under the partial receiver operating curve (%pAUC) as a measure of how quickly methods accumulate true positives at increasing FDR levels until the chosen cutoff at 1% protein entrapment FDR.

B.5 Additional information on runtimes

Method	Runtime (m:ss)
ProteinProphet	8:18
PIA (incl. graph compilation)	5:02
EPIFANY	3:03
Fido	**1:20**
Percolator (incl. PSM SVM)	6:20

Table B.3: Runtimes in m:ss format for the different methods on the almost unfiltered (0.001 posterior PSM) iPRG2016 sample A+B. The best value (= lowest) is highlighted in bold. Times were measured on a MacBook Pro, 2.4 GHz Intel Core i5 (2 cores), 8GB RAM. The dataset consists of 75226 spectra, which after filtering map to 8024 peptides (7384 unique) and 3701 proteins.

Method	Runtime (s)
ProteinProphet	**3**
PIA (incl. graph compilation)	36
EPIFANY	86
Fido	5
Percolator (incl. PSM SVM)	21

Table B.4: Runtimes in seconds for the different methods on the strictly filtered (0.01 PSM FDR) iPRG2016 sample A+B. The best value (= lowest) is highlighted in bold. Times were measured on a MacBook Pro, 2.4 GHz Intel Core i5 (2 cores), 8GB RAM. The dataset consists of 12682 spectra, 4506 peptides (4056 unique) and 1303 proteins.

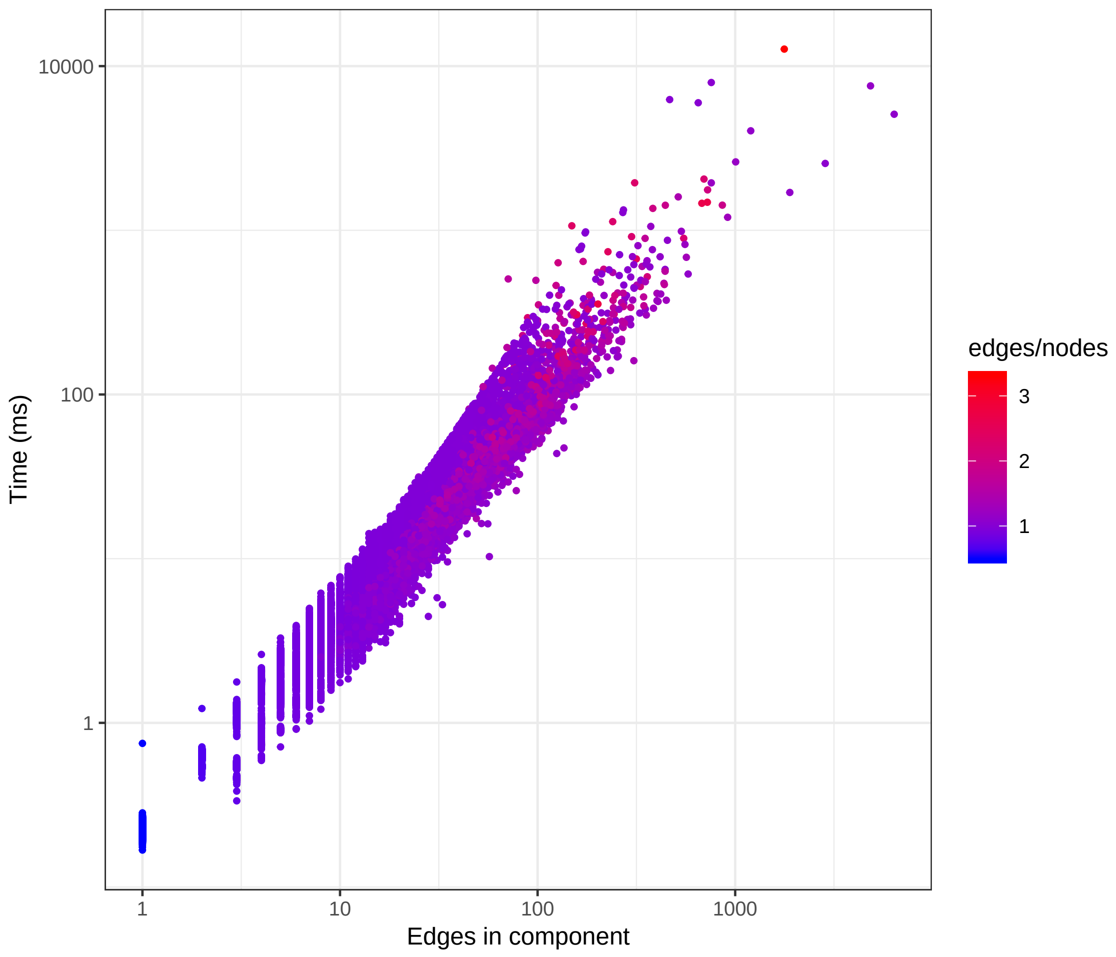


Figure B.21: Runtime of the LBP algorithm per connected component on the large human dataset. The runtimes in milliseconds are plotted against the number of edges in the connected component (after introducing intermediate factors). Colors follow a continuous scale from blue to red to show the number of edges per node as a simplified measure of complexity. Note the logarithmically scaled axes.

C.1 List of identifiers of successfully re-analyzed datasets with ProteomicsLFQ

MSV000085230	PXD006439	PXD010012	PXD013615	PXD018301	PXD023272
PXD000279	PXD007595	PXD012431	PXD013743	PXD018891	PXD023505
PXD002137	PXD007740	PXD012453	PXD014391	PXD019103	PXD023508
PXD002381	PXD007962	PXD012467	PXD014415	PXD019123	PXD023734
PXD002395	PXD008383	PXD012636	PXD014458	PXD019347	PXD024800
PXD003351	PXD008440	PXD012755	PXD015270	PXD020248	PXD025864
PXD004682	PXD008722	PXD012923	PXD015289	PXD020426	PXD027125
PXD005355	PXD008934	PXD012953	PXD015744	PXD020557	
PXD005571	PXD009597	PXD012998	PXD016183	PXD021865	
PXD005733	PXD009630	PXD013150	PXD018230	PXD022915	