PIE - The Protein Inference Engine

Stuart R. Jefferys

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Curriculum of Genetics and Molecular Biology.

Chapel Hill
2011

Approved by:

Morgan C. Giddings

Xian Chen

Jeffrey A. Frelinger

Shawn Gomez

William F. Marzluff

Abstract

STUART R. JEFFERYS: PIE - The Protein Inference Engine.
(Under the direction of Morgan C. Giddings.)

Posttranslational modifications are vital to protein function but are hard to study, especially since several modification isoforms may be present simultaneously. Mass spectrometers are a great tool for investigating modified proteins, but the data they generate are often incomplete, ambiguous, and difficult to interpret. Combining data from multiple experimental techniques provides complementary information. Having both top-down (intact protein mass data) and bottom-up (peptide data) is especially valuable. In the context of background knowledge, combined data is used by human experts to interpret what modifications are present and where they are located. However, this process is arduous and for high-throughput applications needs to be automated.

To explore a data integration methodology based on Markov chain Monte Carlo and simulated annealing, I developed the PIE (Protein Inference Engine). This java application integrates information using a modular approach which allows different types of data to be considered simultaneously and for new data types to be added as needed.

Validation of the PIE was carried out using two realistically imperfect theoretical data sets. The first, based on the L7/L12 ribosomal protein, tested the limits of PIEs performance as intact mass accuracy and peptide coverage decreases. The second set, based on a mix of two modification variants of the H23c Histone protein, tested PIEs ability to handle isoform mixtures and up to eight simultaneous modifications.

The PIE was then applied to analysis of experimental data from an investigation of the modification state of the L7/L12 ribosomal protein. This data consisted of a set of peptides identified as associated with some L7/L12 modification variant and nine intact masses measurements identified as an L7/ L12 modification variant. From this data, PIE was able to make consistent predictions, comparable to expert manual interpretation.

Software, source code, user manuals, and demo projects replicating the analyses described in the following can be downloaded from `http://pie.giddingslab.org/`.

To Sara, for putting up with all my rantings.

Acknowledgments

Without the help and support of colleagues and friends, PIE would never have come to be. I would like to thank the current and former members of the Giddings lab: Morgan Giddings, for the opportunity to work on such a great project, and for the crazy notion that I should try integrating proteomics data using McMC; Mark Holmes for building Proclame, the fuzzy logic based top-down proteomics program I worked on during my initial rotation in Dr Giddings lab which convinced me there was cool stuff to do there; Kevin Ramkissoon, Hsun-Cheng Su, and Eric Hamlett for helping me understanding the practical side of proteomics and for providing the data analyzed herein; Dennis Crenshaw, Chris Maier, Jameson Miller, and Suzy Vasa for a great deal of good programming advice and geek talk; Jainab Khatun, Dongmei Yang, and Brian Risk for many side adventures and their kind ear when nothing was working; and Ashley Lundqvist, for helping to keep the train on the tracks.

I would also like to thank the Genetics and Molecular Biology Curriculum (GMB) and Bioinformatics and Computational Biology (BCB): Bob Duronio, for his support and commitment to the students of GMB; Cara Marlow, and Sausuty Hemereck for hiding the most difficult part of graduate school from me—the administration and paperwork; and Alex Tropsha for championing the cause of Bioinformaticsat UNC.

Of course, I owe a debt of gratitude to the members of my committee: Jeff

Frelinger, Xian Chen, Shawn Gomez, Bill Marzluff and Morgan Giddings (chair). I would also like to thank many of my fellow students who helped both directly and indirectly, especially by providing good excuses to go have a beer up at Top of the Hill: Kate Arrildt, Matthew Berginski, Kwangbom Choi, Damien Croteau-Chonka, John Didion, Jeremy Simon, Jennifer Staab and everyone else. Also important to the success of this project were the many friends who were there when things were hard: Sean Korb, David Nolin, Jenn Skahen, Jason Sullivan, Andrew Sparling and especially Sara Wrenn.

Last but certainly not least I am grateful to the NIH and to UNC's GMB and BCB programs for the funding that supported me, as well as Joe van Gogh and Caffe Driade, for providing the coffee that did likewise.

Preface

The base algorithm used in this work, Markov chain Monte Carlo or McMC (Metropolis, 1953), dates back to the earliest years of computers, it was developed on a large tube-based computer called MANIAC and remains an important way to sample from very complex distributions. The seemingly minor change of adding a cooling strategy changes the algorithm significantly, into simulated annealing or SA (Kirkpatrick, 1983), which allows optimization over very complex functions. One of the main features of McMC/SA is that they are very resistant to complexity scaling, and can be used to find answers to large problems that few other methods can address and that no method can solve. Many of these problems are combinatoric in nature. A trivially small but still surprisingly challenging example is given by an online cartoon: `http://xkcd.com/287/`. I won't give away the answer, but I was able to adapt PIE to solve it with about thirty minutes work.

As with any significant effort, many people have had a hand in helping me create PIE and deserve a big share of the credit—some of the most valiant are mentioned in the Acknowledgements. However, as the final arbiter of what went into this written work, all the blame for any errors belongs to me.

Portions of this work have been published previously. The paper that should be referenced for theoretical and validation details is:

Jefferys, S. R. and Giddings, M. C. (2011) Baking a mass-spectrometry data PIE with McMC and simulated annealing: Predicting protein post-translational modifications from integrated top-down and bottom-up data. Bioinformatics. 27, 844-852.

The paper provides a much more concise summary of the methodology of PIE. Most of chapter 4 (Results) and the first and last parts of chapter 5 (Discussion) are similar or identical to contents in this paper, along with figures 3, 4, 5, and 6 and all of the tables. Some copyrights are held by the publishers, Oxford University Press—used by agreement.

Additionally, a more task-oriented tutorial has been published as chapter in a book on comparative proteomics:

Jefferys, S. R. and Giddings, M. C. (2011) Automated data integration and determination of posttranslational modications with the protein inference engine. Wu, C.H. and Chen, C. (eds) In Bioinformatics for Comparative Proteomics. Chapter 17, 255-290. Springer. Heidelberg, Ger.

The book chapter provides a more concise view of the details of the PIE software described in this thesis. The middle part of chapter 5 (Discussion), Appendix C, (User Manual), and much of Appendix D (Tutorial Walkthrough), is similar or identical to material from the book chapter, along with figures 1,2, 7, 8, and 9. Some copyrights are held by Springer/Kluwer Academic Publishers—used with the kind permission of Springer Science and Business Media.

Besides the explicit elements, other parts of these two previous works are woven together and expanded on throughout this thesis.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AA	amino acid
Acet	an acetylation adduct
Asn	asparagine
Asp	aspartic acid
BU-MS	bottom up mass spectrometry
Da	Dalton
E. Coli	bacterium Escherichia coli; where it matters, strain K12
e.g.	exempli gratia (for example, including)
H23c	canonical human histone protein 2.3c
i.e.	id est (that is; in other words)
k	kilo, applied as prefix to mean multiply 1,000
L7L12	large ribosomal protein 7/12 from E. coli, strain K12
Met	methionine
Meth	a methylation adduct
MS	mass spectrometry
MS/MS	tandem mass spectrometry
McMC	Markov chain Monte Carlo
PCR	polymerase chain reaction
Phos	a phosphorylation adduct
PIE	protein inference engine
ppm	parts per million
PTM	posttranslational modification
SA	simulated annealing

Ser	serine
Thr	threonine
TD-MS	top down mass spectrometry
TDBU-MS	top down/bottom up mass spectrometry

List of Symbols

D	collection of all data used that informs on a proteins PTMs
G	solution space of all guesses, each a modified protein isomer
g	singe guess; one specific protein modification isomer
i	a variable that iterates, usually from 1 to n
M_t	theoretical mass calculated from molecular composition
M_e	experimental mass measured by mass spectrometry
E	error window
n	count of something
m	count of modifications
R	the score ratio for two modification isomers
S	the score of a guess
$S()$	a scoring function

Chapter 1

Introduction

1.1 Understanding Proteins Requires Studying Protein PTMs

Proteins are the main agents of function in a cell and underlie many of the processes that sustain life. They catalyze cellular reactions, transmit and amplify signals in kinase networks, coordinate cellular processes and act as scaffolds to provide cellular structure. Proteins are woven together in a cell to create a complex and dynamic system. One key aspect of that system is the modulation of protein behavior by chemical changes made co- or post-translationally (Seo, 2004; Walsh, 2005). Some of these chemical changes alter the sequence of a protein after its production by the ribosome, removing a number of amino acids from one or both ends. Other changes involve chemical adducts, functional groups that are added to or subtracted from proteins–often by other catalytic proteins that are themselves modified. While modification can occur during translation, all modifications hereafter are referred to as post-translational modifications, or PTMs.

A cell needs to modify proteins with posttranslational adducts for the same reason that an automobile needs gears: without them the operational range of a protein engine is severely limited. If cells relied on fixed proteins, biological reaction

times would be on the order of minutes and proteins would be limited to only the 21 functional groups provided by tRNA-delivered amino acids. Cells use 300+ known PTM adducts (Creasy and Cottrell, 2004) to allow dynamic functional shifting, providing for rapid and flexible responses to changing local conditions. When PTM regulation breaks down, it is much like a broken transmission in a car–proteins become nonfunctional (Banerjee and Gerondakis, 2007; Minamoto et al., 2001; Shi, 2007). Teasing apart where PTMs occur, when they are used, and how they are modulated is of great interest in biological and biomedical research.

PTMs can alter a protein's function in a variety of ways, such as changing protein-protein interactions and blocking binding sites (Gundry 2007), or adding or removing parts of the protein that specify its destination in the cell. For example, in bacteria like Escherichia coli (E. coli), swimming toward food is directed by a chemotactic circuit that senses nutrients in the environment. The process is controlled by a regulator circuit that is modulated by methylation and demethylation, where modification of residues on receptor proteins change their sensitivity to ligands in the environment. This circuit is also modulated by soluble proteins like CheY and CheA which become phosphorylated in response to changing nutrient conditions. The downstream result of these modifications are changes in the behavior of flagellar motors, affecting swimming (Kollmann 2005; Kentner 2006). Without PTMs, an organism would be dramatically limited in its ability to respond quickly to changing environmental conditions.

PTMs also play critical roles in human health and disease. Methylation and acetylation of the histone proteins around which DNA is wound control which genes are expressed and when. Misregulation of histone modification can be extremely deleterious (Shi 2007). Many oncogenic processes involve misregulation of kinases and kinase pathways, including p53 (Minamoto et al., 2001). Immune responses are

also frequently modulated by PTMs through toll-like receptor pathways (Banerjee and Gerondakis, 2007). The details of if, when and how proteins are modified is central to uncovering how disease processes work and determining potential therapeutic actions.

Given the important role of PTMs, a key goal of proteomics research has been to develop approaches and methods that can maintain fragile PTMs during handling, then tease apart the subtle signals that indicate the location and type of PTMs on proteins (Mann, 2003; Thingholm, 2009; Durbin, 2010). Although considerable progress has been made, there remain substantial hurdles to realization of a fully automated approach to identifying PTMs.

1.2 The Challenges of Identifying PTMs

Proteins exhibit diverse chemical properties, and PTMs further increase that diversity. This is useful biologically, but it makes the development of techniques to analyze proteins challenging. Studying PTMs requires examining proteins, but without amplification techniques like the polymerase chain reaction (PCR), proteomics methods are significantly more difficult and indirect than genomic methods. Mass spectrometry (MS) is one of the only methods that has been developed which can universally analyze most proteins to identify their chemical composition in their in vivo form (Domon, 2006).

MS measures molecular masses, and this allows us to obtain information about the molecular composition of proteins. To distinguish between a protein containing the amino acid asparagine (Asn), which weighs in at 114.1038 atomic mass units (often referred to as Daltons or Da), versus one containing aspartic acid (Asp), weighing 115.0886 Da, one might use the information that protein X weighs 0.9 Da

more than protein Y to infer that protein A has an Asp where protein B has an Asn. Similarly, when a protein is modified by the addition of chemical groups such as methyl (adding 14.0269 Da) or phosphoryl (adding 79.9799 Da), one can use the change in mass to infer the presence of the modifications. This is illustrated in Fig. 1, "*Intact Mass Shifts Due to PTMs*".

With the advent of new instrumentation such as orbitrap mass analyzers (Perry et al., 2008) combined with the continued maturation of established techniques like electrospray ionization (Maxwell and Chen, 2008), many mass spectrometers are capable of accuracy better than 10 parts per million (ppm). That means a 50,000 Da protein (50 kDa) can be measured within ± 0.25 Da, easily enough to use a mass shift to differentiate between many distinct modification states (isoforms) of a protein.

However, in practice PTM analysis is not simple. Even putting aside sample preparations issues (Fang et al., 2010), analytical difficulties remain, including:

- *Achieving sufficient accuracy to determine PTMs for large proteins.* In many cases, the accuracy of a mass measurement is relative to the size of the protein (as in 10 ppm). For larger proteins, the error is proportionally larger.

- *Working with obstinate, insoluble proteins.* Intact proteins are often quite hard to manipulate prior to and during mass spectrometry. Some dislike staying in solution, and even when they can be kept in solution, they may not ionize or fly well in the mass spectrometer (Albrethsen 2007; Mirzaei and Regnier, 2006).

- *Multiple combinations of PTMs can give the same or nearly the same mass shift.* Often, more than one modification is present, and the mass of three methyl additions fall very close in mass to a single acetyl addition (within

0.0434 Da). Different modifications or modifications sets with the same mass are said to be isobaric.

- *Determining precise positioning of PTMs.* Measuring the mass of the whole protein to determine what modifications are present does not tell us where any of the modifications are located, and a different location of the same modifications can mean the difference between up and down regulation (Shi, 2007).

Several approaches have been devised that address these challenges, including bottom-up, top-down, and combined methods.

1.3 Bottom-Up, Top-Down, and Combined MS Methods

Due to the many challenges of PTM analysis, two different methodologies have been developed, top-down mass spectrometry (TD-MS) and bottom-up mass spectrometry (BU-MS). Neither are fully satisfactory although BU-MS methods are perhaps easier or at least cheaper, and have come to dominate the literature. A summary of TD-MS and BU-MS methods that focuses on their drawbacks follows; more comprehensive reviews are easily found in the literature (e.g. Bogdanov and Smith, 2005; Domon and Aebersold, 2006; Yates et al., 2009).

Bottom-up MS uses a divide-and-conquer strategy that reduces proteins to constituent, short peptides that are more readily analyzed. The procedure begins by digesting a protein into peptides, generally with an enzyme such as trypsin that cleaves at predictable sites. These peptides are then separated based on distinct chromatographic and/or chemical properties and analyzed by MS to infer PTMs.

Bottom-up analysis simplifies the analysis of PTMs by producing smaller, more

accurately measurable peptides and by decoupling modifications that fall on different peptides. However, although reduced in scope, the basic problem of determining modification locations and resolving isobaric masses remains at the individual peptide level. To overcome this, another round of divide-and-conquer can be applied, breaking a peptide into a set of constituent fragments. Given enough fragments, the amino acid sequence of the peptide can be reconstructed. This process requires two successive mass analysis steps (ones to select a peptide mass, another to fragment and analyze the peptide), hence the name tandem mass spectrometry or MS/MS. The resulting tandem spectra can be used to precisely locate a PTM on a given residue in the peptide, since it will cause a discernible shift at that site. This is shown in Fig. 2, "*Tandem Mass Spectrometry (MS/MS) Locates PTMs*". With appropriate software, the peptide mass and/or its MS/MS spectrum can be used to determine what kind of PTM was present, and often, which specific residue it was present on.

To interpret bottom up data, most software relies on alignment algorithms to compare experimentally acquired peptide and MS/MS spectra against a database of known spectra. Interpreting the difference allows identification of PTMs. Examples of this type of software include cross-correlation (Yates et al., 1995), Mascot (Perkins et al., 1999), TANDEM (Craig and Beavis, 2004), and InsPecT (Tanner et al,. 2005). More recent work tackles the problem head on, performing de novo interpretation of MS/MS spectra, e.g. Spectral Dictionary (Kim et al., 2009) or an Integer programming based algorithm, PILOT_PTM (Baliban et al., 2010).

Although bottom-up analysis with MS/MS is quite a powerful technique, it is not a complete solution. Two of the most significant problems involve decoupling of PTMs by the protein digestion step and the inability to detect all or even the majority of resulting peptide fragments.

If there are multiple PTM isoforms of a protein present before digestion, determining which PTM goes where can be difficult once the protein is digested. Cleavage into peptides can decouple PTMs if they reside on separate peptides. For example, if a protein has two phosphorylation sites, each on a separate peptide, it is not possible using a bottom-up strategy to tell the difference between a sample containing a mix of an unmodified and a doubly modified isoform variant, versus a sample containing a mix of two singly phosphorylated isoforms (each phosphorylated at one of the two sites). The digestion step used by the bottom-up approach converts either protein mix into identical peptide sets, each containing a modified and an unmodified version of each phosphorylation site.

Another challenge with the bottom-up MS/MS approach is the high frequency of missing peptides. Full protein coverage requires identification of a complete set overlapping or abutting peptides. This is almost never achieved due to various factors including peptide concentration, solubility, and MS ionization characteristics. If a peptide is not observed, all information regarding its PTMs is lost.

Even when peptides are detected, it may not be possible to investigate the MS/MS spectra for all peptides. It can also be difficult to definitively identify the site and type of PTM within an MS/MS fragmentation spectrum. Although there are significant success stories involving bottom-up methods, the difficulties have left room for developing a separate approach, top-down MS.

Top down MS starts with loading the full intact protein directly onto the mass measurement, and then take it a few steps further by fragmenting the intact protein within the mass spectrometer by one of a variety of methods, then measuring the pieces. This is also a divide-and-conquer approach, and in some cases, the pieces may themselves be isolated and further fragmented, leading to a process of MS(n), where n reflects the number of successive fragmentation steps (e.g. Zubarev et al.,

2002; Kelleher et al, 1999). For proteins whose solubility and size allow loading and detection by MS/MS, top-down mass spectrometry has great potential as a faster and more complete way of analyzing PTMs on proteins.

However, TD-MS faces its own challenges, such as the difficulty of isolating and analyzing intact proteins, and the difficulty of interpreting the complex top-down spectra (Siuti and Kelleher 2007). The top-down method is typically performed using electrospray ionization as the ion source, which produces ion species in multiple charge states. The process of deconvoluting the multiple species and their multiple charge states is a long standing problem and is difficult to solve (Ferrige et al., 1992). PTMs complicate this further.

TDBU-MS seeks to combine TD-MS and BU-MS methods to address their separate failings and obtain a more complete analysis. Recent work by multiple groups has seen a variety of hybrid approaches (Yu et al., 2005; VerBerkmoes et al., 2002; Strader et al., 2004). With a combined TDBU-MS strategy, bottom-up data contributes definitive protein identities, along with a partial map of specific modification sites and types, while the top-down data provides insight on the overall state of the protein (e.g. is there one phosphorylation or two at any given time on the protein?). As TD-MS data provides constraints on the possible total number of modifications, adding it to BU-MS data helps resolve the PTM decoupling and missing peptide problems.

But TDBU-MS has its problems too: integrating the data from disparate mass spectrometry experiments and approaches into a cohesive picture of the protein's original state. This is not a trivial problem. The experiences of colleagues trying to put together intact mass, peptide, and MS/MS data for a study of ribosome modification (Kevin Rankisoom - unpublished data) revealed the difficulty of manual integration.

Due to incomplete bottom-up information and multiple equivalent isobaric PTM configurations, there are many possible interpretations for any data set. Each measurement holds a piece of information about the protein's state, but it is usually incomplete. And worse, sometimes data conflicts with itself. For example, if two isoforms of a protein are present, one with a methylated residue and one without, some bottom up peptides covering the methylation site indicate the presence of a methylation, other peptides covering the same site suggest it is unmodified.

Nevertheless, with substantial human effort it is often possible to produce a clear, good answer. Generally these answers are not absolute, but an expert can usually provide a concordant argument for their choices. This kind of reasoning is difficult to turn into an effective and practical computer algorithm. Yet the flood of new mass spectrometry data does not allow for human experts to examine every output, necessitating computational tools that make the process significantly more efficient.

Because the data integration problem is so great, recent effort has focused on solving this problem. While various scripts have been created to search constrained subsets of modification possibilities, tools to aid in the automation of this process are just now starting to emerge, such as PTMSearchPlus (Kertesz et al., 2009), which performs bounded search constrained by the statistics of the most likely numbers of modifications to occur, and (Durbin, 2010), where bottom up and top down data were combined using a pipeline composed of multiple steps, including the ability to identify the location of 3 or so important modifications. Though these approaches provide integrated solutions utilizing a pre-specified set of data types, adding additional data not part of the original design or pipeline is difficult. The basic challenge of integrating arbitrary data in a way that minimizes hard bounds regarding the PTM scenarios that can be considered is a difficult problem. A human

expert uses many other data sources (expressed as prior knowledge) in determining a final solution, and has the capability to consider novel situations, outside of the bounds of a constrained search. It is this task PIE (Jefferys and Giddings, 2011b) was created to address.

1.4 PIE Automates Data Integration for TDBU-MS, and More

The PIE is designed to rapidly and automatically integrate disparate types of proteomic measurements into a conclusive picture of the modification state of the protein. It is highly modular, with each module allowing it to incorporate a distinct type of information. Presently there are modules that use TD-MS data (in the form of intact mass measurements), BU-MS peptide, and MS/MS data. It also has multiple modules incorporating a wide variety of prior-type data such as residue-specific probabilities of various modification types and expert knowledge from specialized PTM site predictions programs like SignalP (Bendtsen et al., 2004), TermiNator (Frottin et al., 2006), NetPhos (Blom et al., 1999), and Sulfinator (Monigatti et al., 2002). PIE can readily accommodate conflicting information; if after weighing all the data there is not enough information to select a single best solution (e.g. there are multiple isobaric isoforms), the program will output multiple high-scoring solutions. By using multiple types of data and overlapping constraints, PIE finds consistent answers even when peptides are missing and will identify and localize modification to regions of the protein not covered by peptide data if such an assignment is supported by intact mass and prior data. Furthermore, information about why PIE has come to the answers it has is available, it the form of how PIE weights each type of data in support or disagreement with each answer given.

Chapter 2

Theory

2.1 Finding the Solutions to Problems

Inspiration for PIE's solution to the data integration problem comes from Plato's allegory of the cave (Plato, 1968). As cave-dwellers, humans are forever barred from direct experience of the real world, limited to watching only the shadows of a true outside reality that is projected onto the cave walls. It is only possible to imagine what truth actually is using the shadows observed. Making inferences based on shadows is all that can be done, and guesses about the underlying, unobservable reality (that can be used to reproduce observable shadows) are the only truth. Applied to the context of a protein and its PTMs, the shadows are the data revealed through the light of experiments such as TD-MS and BU-MS. PIE is trying to guess the specific pattern of PTMs that is most probably the underlying truth that resulted in the data shadows observed.

One thing that is not made clear in the allegory is how to go about guessing at reality. Trying to pick a wining lottery ticket requires guessing a number that will either win or loose, and if the game is fair, there is no way to determine ahead of time which guesses are more likely to win. Usually, however, knowing that

one guess produces a given set of results helps determine what else will happen. Trying to guess a number between 1 and 1000 is easy given feedback that a guess is too high, to low, or correct. If the first guess g_1 is $n/2$ (with n = the largest possible guess), and each successive guess g_i is changed by $\pm(g_{i-1})/2$, with g_{i-1} the previous guess. It only takes about $\log_2(n)$ guesses to get the right answer. This is at most 11 guesses for $n = 1000$. Not only is feedback necessary, there must be a relationship between between possible results. In order to use a guess and refine solution method rather than trial and error, their must be structure in the way possible guesses are organized. If the possible guesses were randomly jumbled together instead of being aligned in monotonic increasing order from 1 to 1000 a trial and error lottery would be the only way to get to an answer. In the cave, there must be a way to structure different guesses about reality such that similar guesses have similar shadows. Usually, this means arranging the collection of all possible guesses into a "space", where distances between solutions can be defined and nearby answers are similar.

2.2 The Problem of Determining Protein PTMs Given Data

PIE is really just a sophisticated version of guess what modified protein I'm thinking of. A collection of data, D, reflects observations about the PTM state of a protein. Another collection, G, represents all guesses that might be made about the underlying truth–the protein's PTM configuration. The goal is then to identify the guess $g \in G$ most consistent with the data. Evaluating candidates for the best guess is done using a scoring function $S(G|D)$. This assigns to each guess g a score based on the available data. In this formulation truth is then the highest scoring guess:

$$\text{Best guess for truth} = \text{argmax}_G(\, S(G|D)\,) \tag{2.1}$$

To apply this description of PTM inference and build a guess and refine prediction engine for PTMs, four components are needed: A space of solutions or guesses G, a bunch of data D, a scoring function from S onto D, and a method for finding the maximum scoring $g \in G$. The rest of this section describes the requirements for these components.

2.2.1 Defining the solution space G

The number of possible guesses in set G is rather large. For just 10 different adduct modification and a short protein of 100 residues, G contains a googol (10^{100}) possible answers, each a different modification state. This is much, much larger that the age of the universe, which is only around 10^{30} *pico*seconds! (Bolte and Hogan, 2002). It is well and truly impossible to check each possible scenario to find the best one. In order to use a guess and refine solution method rather than trial and error, G must be organized into a multi-dimensional space where similarity is defined using an abstract model of a modified protein.

For a protein of length n, each amino acid has a position from 1 to n and can have one of $m + 1$ adduct modifications. The plus one allows for the null modification, indicating an unmodified amino acid. Additionally, left (n-terminal) and right (c-terminal) cleavage positions are specified as the first and last amino acid not cleaved. This creates a solution space of $n + 2$ dimensions, where each dimension has as discrete values corresponding to each possible modification type. The search space is visualized in Fig. 3, "*Solution Space Representation*". In this solution space, a segmented line from left to right represents a modified version of

the protein. The same line is always the same modifications, PTM patterns that are similar result in similar lines, and PTM patterns that are very different result in different lines. This solution space G allows a rough continuity and distance definition for the scoring function $S()$ over G; PTM isoforms that are close together are similar and hence have similar goodness. Together G and $S()$ will create a functional landscape to hunt around in for the best scoring guess.

One aspect of the cave allegory that may not be obvious is the complete reliance on the imagination of the cave dwellers to be able to guess at the truth. Anything that can not be conceived will never be guessed, regardless of the data. Once imagined and played with, theories like the conservation of energy or evolution by natural selection become second nature and are taught in high-school or earlier. But for centuries they remained undiscovered. Human creativity in imagining truth given only shadows is part of why the cave analogy seems so apt.

Unfortunately, computers are very bad at creative reasoning. They are incapable of rising above the specific bounds set for a problem, and PIE is no different. This search space G defines the only kind of answer PIE can come up with. It is surprisingly creative within these bounds, in some cases coming up with unexpected correct answers, but the solution space imposes several constraints on the kinds of problems that can be addressed with PIE.

- The underlying canonical protein sequence must be known.
- PIE works with one protein at a time, and all experimental data to be used must be associated with that protein.
- The list of adduct modifications to consider must be known. Only data about fixed modifications can thus be processed, variable glycosylation or lipid modifications are not allowed for.

- Each adduct modification is independently assigned to one amino acid. Correlation between modifications must be provided as explicit additional data.

- Each amino acid of the protein is allowed only one adduct modification. Representing more than one modification on an amino acid requires using a single adduct that stands in for a modification combination like dimethylation or trimethylation.

These constraints are acceptable for some kinds of problems, but not for others. Chapter 4 gives the results obtained using PIE to address one such problem, and Chapter 5 discusses these more fully. However, these constraints result only from the choices made to construct G. The process of guess and refine applies to problem of selecting a good G as well; the first guess can be refined and then used in other versions or implementations of PIE targeted at different problems. Although the G described above represents a minimal description of a modified protein, it is easy to add complexity. For instance, a model that also allowed each AA to vary over the 20 amino acids would partially abrogate the first constraint, but would double the number of dimensions in the solution space. Greatly expanding the list of modifications to include all possible mutations (like Ser $\Rightarrow$ Thr), would likewise address the first constraint at the cost of increasing the size of the search space. The search space G is already huge, and with every dimension added or increased significantly in size, it becomes harder to search, taking longer and requiring more data to search successfully. Using a minimal G allows the simplest search that works with the least data.

2.2.2 Finding $\text{argmax}_g(S)$ using McMC

The ultimate goal is to find the truth, the modified protein variant underlying the data. Given the solution space G, some data D, and an as yet undefined function $S(G|D)$, to find the truth requires finding $\text{argmax}_g\, S(g|D)$. Metropolis Markov chain Monte Carlo (McMC) (Metropolis et al., 1953), is a heuristic method for sampling from a solution space such as G using a guided random walk. The walk is guided by a ratio R computed using the scoring function $S()$ applied to two neighboring points in G: g_c, the currently point, and g_p, a neighboring solution selected and proposed randomly as a possible next step.

$$R = \frac{S(g_c|D)}{S(g_p|D)} \tag{2.2}$$

If the new guess g_p has a higher score than g_c (when $R >= 1$), then g_p is made the new current point, the next step in the guided walk. If g_p is a worse guess (when $R < 1$) a step to g_p is made only with probability R. If g_p is not made the next current point, then g remains the current point g_c (with probability $1 - R$). Walks start at a randomly chosen point in G and consist of step after step taken by repeating this propose–evaluate–step cycle over and over. After some time, the walk is stopped and the current point is reported. If the walk is long enough, and the space walked over is reasonable, each point will be sampled from the landscape exactly as often as its normalized probability:

$$P(g) = \frac{S(g|D)}{\sum_i S(g_i|D)} \tag{2.3}$$

From the perspective of trying to identify the the modifications that are most consistent with the data (the ones $S()$ gives the highest score too), the walk spends the most time on the mountains, the highest scoring (best) points in the space but

will occasionally traverse the valleys to avoid being stuck at a local maxima.

Unfortunately, the near infinite number of low scoring solutions drown out the few high scoring ones, such that the best solution is reported infrequently if at all. Simulated Annealing (SA) (Kirkpatrick et al., 1983) is a modified version of McMC that focuses the random walk so that it only reports the best answers, converting it from a sampling algorithm into an optimization algorithm. Simulated annealing modifies the McMC walk when $R < 1$ by scaling R by a coefficient that decreases gradually with each step, so that near the end of the run the coefficient approaches zero. This has the effect of gradually bounding the McMC walk, preventing it from crossing ever shallower valleys, until at the end it can only go uphill.

If run long enough, SA will always converge to the highest scoring answer, but it is difficult to determine in advance how long that will take. One way to address this is to run the algorithm repeatedly to sample from the space of solutions, providing an empirical distribution showing the frequency with which a given answer is obtained. If run long enough, every answer will be the same, the maximum answer. As this run-length is approached, the maxima will be found more and more frequently. To be efficient, a specific convergence recipe is used when running PIE to determine the best answer; see Section 3.4, Determining the Convergence Length for details.

A subtle detail of the cave allegory that may not be appreciated is the inherent fuzziness of answers. Although the highest scoring answer is the best guess at truth (given the data and scoring models), there may be other guesses that score nearly as well. If the data provided is not perfect and complete, all truths within some range of scores are essentially indistinguishable given the data available, so picking one of them is incorrect. Furthermore, the truth may not be a pure single point in the solution space, but may represent one or more points simultaneously.

Repeated sampling of sub-optimal answers addresses these issues by providing

a way to distinguish between data that supports a single answer and data that consistently supports several different solutions. Sampling sub-optimal answers is performed by PIE through SA walk at run lengths that finds the best score only sometimes; when it does not, a sub-optimal answer is reported. This does require knowing what the optimal answer is, so that must be determined first. Due to the simulated annealing algorithm driving the walk, the sub-optimal answers are still high-scoring answers. The longer the run-time, the more optimal the sub-optimal answers will be. To be efficient, a specific profile recipe is used when running PIE to profile sub-optimal answers; see 3.5, Obtaining a Complete PTM Answer Profile, for more details.

2.2.3 The data D

In order to specify a flexible scoring function compatible with McMC search, a function with the following properties is required: it must be defined and non-negative for all possible guesses, it must model the relationship between data and guesses such that better supported guesses have higher scores, and the ratio between the scores of any two guesses should reflect the relative support for those guesses given the data.

To be flexible, the data D is assumed to consist of some varying combination of up to k different data types. $D = \{d_1\} \cup \{d_2\} \cup \ldots \cup \{d_k\}$. Each type is a cohesive collection of data describing a protein's modifications. For example, d_1 might be an intact mass measurement, d_2 a set of peptide masses, d_3 a set of MS/MS derived sequence data, d_4 a summary of how common or rare different modifications are, etc. Defining the complete joint probability distribution $P(G|D)$ for even simple data sets is difficult, and it is certainly incompatible with the desire to allow easy extension of PIE to include additional data types. This requires

the simplifying assumption that each data type is independent. As with Naive Bayesian classifiers, the error this introduces at least partially cancel thought the use of multiple data types. (Zhang, 2004). In other words, rather than squeezing ever drop of information out of a small amount of data, a lot of data is used, albeit more wastefully. Independence allows expressing $P(G|D)$ as the product of the individual probabilities for each data type:

$$P(G|D) = P_1(G|d_1) \cdot P_2(G|d_2) \cdot \ldots \cdot P_k(G|d_k). \tag{2.4}$$

2.2.4 The scoring function $S(G|D)$

During its search, McMC only evaluates answers by the ratios of their probabilities, not their absolute probability of occurrence. This allows us to take a substantial shortcut by representing prior data types through a non-normalized scoring function. Expressed for a single guess $g \in G$ this is:

$$S(g|D) = S_1(g|d_1) \cdot S_2(g|d_2) \cdot \ldots \cdot S_k(g|d_k). \tag{2.5}$$

The requirements for each factor in this scoring function are the same as those first outlined, but applied to each type separately. Arbitrary new data types can be added in the future by developing and adding new scoring terms that meets these requirements. The data types and associated scoring terms provided by PIE are summarized in Table 1, "*Summary of Scoring Functions*", and in the following chapter on methods. More detailed is provided in C.4, Scoring Module Reference.

Chapter 3

Methods

3.1 PIE Implementation

This section describes in detail the implementation of the methodology outlined in the introduction.

PIE is written in Java 1.5. Data interpretation and scoring are modularized with each type of data evaluated by a separate scoring module. This provides flexibility to add new data types in the future through additional modules. All complex input data is read from simple table-like delimited text files; output is similarly presented. Rather than complicated command-line parameters, control information (along with some simple input data) is provided via a standard java properties file. The program can be download from `http://pie.giddingslab.org/`.

PIE takes as input some data D, a collection of information gathered in mass spectrometry experiments (intact mass, tandem MS/MS, etc) along with various types of prior data and expected distributions. Also included is the target protein sequence, composed of n amino acids $a_1 \ldots a_n$. The sequence of the protein must be specified by the user. Protein identity and sequence is readily determined by one of many algorithms that can match MS/MS data to a protein sequence database

(Perkins et al., 1999; Wisz et al., 2004)). PIE finds the guess g, some modification isomer of the target protein, that has the highest-scoring fit given the data D.

Each guess at a protein variant g is composed of a set of modifications mod_i, with i corresponding to the amino acid position in the protein. The most frequent modification is then none. A guess g also has two cleavage position indicators for potential n- or c- terminal cleavages, one at position $1 \leq \text{term}_n \leq n$ and one at position $\text{term}_n \leq \text{term}_c \leq n$. (See Fig. 3, "*Solution Space Representation*").

The McMC walk will proceed stepwise through the space of guesses G by proposing PTM changes to the current guess g, distributed as follows: 40% of proposals will set the modification at a randomly selected position to null; 10% of proposals will change a randomly selected position to a non-null adduct; 20% of proposals will swap the modifications at two randomly selected positions; and 30% of the proposals will change the c-terminal or n-terminal cleavage marker by ± 1 (7.5% for each possible change). Using a multi-jump approach for improved crossing of wide valleys (Gilks, 1996), more than one change may be applied before scoring, where the probability of n changes is $n/2$ (e.g. The probability of 2 changes is 0.25).

The algorithm applies the proposals in a random walk across guesses g starting from an unmodified version of the protein (all modifications are none). The first 2000 steps proceed without scoring to provide a random starting point. For each step beyond 2000 up to a specified number of steps (the run-length), the scoring function $S(g)$ will be used to calculate the ratio R between the score of the current protein guess $S(g_c)$ and the score of the guess after one proposed step, $S(g_p); R = S(g_c)/S(g_p)$.

If $R \geq 1$, the new guess g_p is kept and becomes the current guess g_c for the next step. Otherwise, if $R < 1, g_p$ may still be kept as the next step g_c, but only with probability $R \cdot z$. If the proposal is rejected then the next step remains as

the current g. The factor z is the simulated annealing factor, which starts at 1 but is reduced at each step linearly by 1/run-length at each step. Decreasing the probability of accepting guesses is referred to as "cooling". Other cooling strategies are possible (Nourani and Andresen, 1998).

The run length and the number of guesses to provide using that length are provided to PIE along with other parameters in the properties file. PIE will run and generate the number of answers specified. For each answer described, PIE will report the total score for that guess and the partial scores for each data type scoring module.

3.2 Scoring Modules

Each data type is modeled by a separate scoring module. Scoring modules are not normalized, and are treated independently. Calculations are optimized for speed rather than for detailed modeling accuracy, as greater flexibility is obtained by using several small fast modules vs one big slow one. Either average or mono-isotopic mass can be used, most abundant isotopic mass measurements are approximated by average mass. Scoring is always against a modified variant of a single known protein, with one modification (usually null) per amino acid. Only modifications for which data has been provided will be considered.

Besides experimental data types, prior-type models are defined that score guesses g against data D consisting of averaged expectations or beliefs about modified proteins. Several of these functions use parameters. Where parameter values are not directly suggested by the data (such as the error parameter in the intact mass), values are selected that reflect the intent of the model to differentiate two adjacent answers, using 0.9 as little but noticeably different, and 0.5 as significantly different.

These were not formally determined by a variation of parameters approach, although multiple runs of PIE have been performed to test the behavior of each model. Using 0.5 means one such measured difference halves the score. Using 0.9 means 7 such differences are needed ($0.9^7 = 0.48$).

3.2.1 Intact mass model S_1

S_1 scores a guess g using data D_1, consisting of an experimentally determined intact mass M_e (known to be a variant of the protein target) and the estimated absolute mass error of the instrument, E. To evaluate a guess g, its total theoretical mass M_t is calculated and used for comparison to M_e in a center and spread model:

$$S_1(g, D_1) = \frac{1}{|M_e - M_t| + E} \tag{3.1}$$

This is defined and greater than 0 for all g, provided guesses alway have at least one AA (true as implemented) and $E > 0$ (Validated on input). The best score is obtained when the guess and intact mass are equal, but gets progressively worse the more the guess and experimental data differ, dropping to half its maximum value at a difference equal to E.

3.2.2 MS/MS sequence model S_2

S_2 scores g against D_2, peptide sequence data where $i = 1 \ldots n$ peptides have been experimentally detected and for which sequence and potential modifications have been determined, e.g. by programs such as FindMod (Wilkins et al., 1999). To evaluate a guess, the alignment to the target protein, the number of amino acids (AA_i), and the number of modifications (m_i) that match identically between the

aligned peptide i, and the guess are used in an exponential model:

$$S_2(g, D_2) = \prod_i (2 \cdot AA_i \cdot 2 \cdot m_i + 1) \tag{3.2}$$

The one is added to avoid singularities when a guess matches no AA or modification for any peptide, required to makie $S_2 > 0$ for all guesses. It is a negligible part of the score of even short peptides ($2^3 \cdot 2^3 = 64$). Any small number would suffice. The more sequence data that matches, the better the score, with a maximum being any guess that matches all peptide data. The base of 2 is chosen so that for each AA and each modification that does not match between sequence data and the guess, the score is halved.

3.2.3 Peptide mass model S_3

S_3 scores g using D_3, peptide mass data. D_3 consists of n peptides that have been experimentally detected and matched by their experimental masses $M_{e,i}$ to a region of the target protein, e.g. by a software such as Mascot (Perkins et al., 1999) or GFS (Wisz et al., 2004). Matching takes place within some estimated absolute mass error of the instrument, E. Each peptide i has a mass $M_{e,i}$ and an alignment to the target protein. To evaluate g, the theoretical mass of the matching region, $M_{t,i}$, is calculated. Each peptide is scored independently and then combined:

$$S_3(g, D_3) = \prod_i \frac{1}{|M_{e,i} - M_{t,i}| + E} \tag{3.3}$$

This is the same as the intact mass model (3.2.1), except each peptide is measured separately and combined into a whole. It meets the requirements for a scoring function in the same way as the intact mass module does.

Two other scoring models that were considered. A summation model added

together terms for each peptide using the exact mass of the matching peptide predicted $M_{m,i}$ weighted by some transform T_i of the match score returned by the matching program for that peptide. $M_{m,i}$ is used instead of the experimentally measured mass as error should already be accounted for in the match score.

$$S(g, D) = \sum_i \frac{T_i}{|M_{m,i} - M_{t,i}| + E}$$

This proved unsatisfactory in testing, possibly because the match scores for modified peptides are generally lower than unmodified ones, resulting in a bias against modified peptides. Another scoring model attempted to use an exponential model based on error counting, similar to that used in the MS/MS sequence model (3.2.2), except counting modifications as matching at all positions each peptide covers. A summation based version of this model worked poorly, possibly due to scaling issues caused by mixing power-based scoring for each peptide term with a total score that balances combinations of peptides as a linear combination. A product based error-counting model was not attempted as the mass-based product model provided a simpler solution. However, problems with this model still remain; Chapter 4, Results, and Chapter 5, Discussion address in more details what PIE gets wrong.

3.2.4 Adduct frequency model S_4

S_4 scores g against the prior knowledge that some modifications are more common than others. Data on the relative frequency of various (non-null) adducts was taken from (Lee et al., 2006), which is in the form of counts, c_{mod} of modification type *mod*, found when scanning protein databases such as uniProt (`http://www.uniprot.org`). Due to the large difference between the probabilities of common modifications

like phosphorylation and rare modifications like selenocystine, log scaling is used. To evaluate a guess, each position i that has a non-null modification (of type *mod*), is given a relative frequency $f_{mod,i} = \log(c_{mod,i}) / \log(\max(c_{mod,i}))$, while $f_{mod,i} = 1$ for non-modified positions. These $f_{mod,i}$ are then multiplied to give a total likelihood score for the observed modification set. To improve the independence of this scoring function from the Modification count model (3.2.6), the likelihood score for each modification i is further scaled from $[0 \ldots 1]$ to the range $[0.9 \ldots 1]$, using $\alpha = 0.9$, a parameter related to how often modifications are expected on average.

$$
\begin{aligned}
S_4(g, D_4, \alpha = 0.9) &= \prod_i f_{mod,i} \cdot (1 - \alpha) + \alpha, \quad \text{where} \\
f_{mod,i} &= \frac{\log(c_{mod,i})}{\log(\max(c_{mod,i}))}
\end{aligned}
\tag{3.4}
$$

$S_4 > 0$ for all guesses, and the more the modifications on a guess match the frequency with which modifications are found that average, the better it will score. Even the most unlikely one only reduces the score by 0.9 (the scaling factor α). But comparing any two modification sets with the same number of modifications, the one consisting of all phosphorylations (the most common *mod*) will score highest and the one with all selenocystines (the least common *mod*) will score the lowest.

3.2.5 Adduct location model S_5

S_5 is similar to S_4, it scores g against the prior knowledge that the probability of a given amino acid having a given adduct varies. Data on the relative frequency of amino acid preferences for non-null adducts were taken from (Lee et al., 2006) , in the form of a matrix of counts, $c_{mod,aa}$, of each modification type *mod* and each amino acid *aa*. Due to the large variability in counts, log scaling is used–

as in S_4. To evaluate g, each position i with a non-null modification, *mod* (on the amino acid *aa* at that position) is assigned a relative frequency $f_{mod,aa,i} = \log(c_{mod,aa,i})/(\log\max(c_{mod,aa,i}))$, where $f_{mod,aa,i} = 1$ for unmodified positions. The $f_{mod,aa,i}$ are multiplied to give a total likelihood score for the observed modification set. As in S_4, a scaling parameter $0 < \alpha < 1$ is used.

$$\begin{aligned} S_5(g, D_5, \alpha = 0.5) &= \prod_i f_{mod,aa,i} \cdot (1-\alpha) + \alpha, \quad \text{where} \\ f_{mod,aa,i} &= \frac{\log(c_{mod,aa,i})}{\log(\max(c_{mod,aa,i}))} \end{aligned} \tag{3.5}$$

This is the same model as in $S4$, and is likewise defined everywhere. However, here the frequency of counts is normalized relative to the most-likely amino acid, so every modification has at least one relatively high-scoring location to put it. This reduces the coupling to the total number of amino acids, so a scaling factor α of 0.5 is chosen to indicate that putting an amino acid on the worst choice, rather than the best choice represents an error and reduces the score by 1/2. An additional impossible factor is used to indicate a modification is placed on a modification it should never be found on, 0.001 is used, as actual zeros are singular points. This is approximately equal to 10 other 1/2 scale errors. Theoretically PIE can still suggest such answers, but only when no other better answers can be found.

3.2.6 Modification count model S_6

S_6 scores g against the belief that a protein is probably unmodified, but a moderate number of modifications are reasonable. This is a center and spread model, similar to the intact mass model (3.2.1). Center c_e is the number of adducts expected, c_t is the number in g, and the spread $\pm E$ gives the expected range.

$$S_6(g, D) = \frac{1}{|c_e - c_t| + E} \tag{3.6}$$

This works the same as S_1; all guesses where the number of non-null modifications is within the range $c_e \pm E$ having a score between the max (if it has exactly c_e modifications) and 1/2 the max (if it has $c_e + E$ modifications, for instance). The parameter E should be set based on the expected rate of modification given the target (i.e. larger for histones), but in the absence of data, a moderate modification rate (say 1 in 10 to 1 in 20 AA) can be used to estimate E and the modification rate can be set to 0, to encourage PIE to find simpler (fewer modifications) answers with higher priority.

3.2.7 N-term cleavage model S_7

S_7 constrains cleavages. It tests g against the belief that the most common state for a modified protein is uncleaved, that N-terminal loss is the next most common (e.g. Met loss), and longer cleavages are exponentially less common This is a geometric model based on the number of amino acids (n) cleaved from the n-terminus of g.

$$S_7(g, a = 0.9, b = 0.8) = a \cdot b^{n-1} \tag{3.7}$$

$S_7 > 0$ for all guesses $n \neq 0$, and is defined as 1 for n=0. It decreases in score such that cleavage of more than 3 AA reduce the score by about half. Better cleavage models are easy to imagine, but as a simple constraint allowing for limited n-terminal cleavage, it works well enough. The switch from 0.9 to 0.8 is used to differentiate the common 1-AA cleavage (Met loss) from less common additional cleavages.

3.2.8 C-term cleavage model S_8

As S_7 Except this scores cleavages from the C-terminus, and there is no difference between cleavage of the first and following AAs.

$$S_8(g, a = 0.8, b = 0.8) = a \cdot b^{n-1} \tag{3.8}$$

3.2.9 Rules model S_9

S_9 scores g against beliefs expressed as simple rules. For all guesses meeting a rule criteria, a constant factor R is applied for each of n instances of some defined pattern in g. The total factor for one such rule R is then R^n. Multiple (i) such rules can be applied, with the total score based on the product $R_1 \cdot R_2 \ldots R_i$.

$$S_9(g, D_9) = \prod_i {R_i}^{n_i} \tag{3.9}$$

. Two rules are implemented:

Amidation and deamidation pair penalty, R_1

R_1 penalizes each pair of amidation and deamidation by a factor of 0.5.

$$\begin{aligned} R_1 &= 0.5^n, \quad \text{where} \\ n &= \text{ count of amidation/deamidation pairs} \end{aligned} \tag{3.10}$$

A complete $m \times m$ map of mutual modifications scores could be added as its own scoring module, but amidation and deamidation are arguably the most important pair for PIE since they are a net 0 D mass change and will escape constraints by

intact or peptide experimental mass data. The 0.5 scales belief that even one such pair counts as a mistake.

N-terminal acetylation bonus, R_2

R_2 doubles the probability of acetylation on an N-terminus even if the residue at that site would not ordinarily be acetylated.

$$\begin{aligned} R_2 = 1 \quad & \text{if n-termini is un-acetylated;} \\ = 2 \quad & \text{if n-termini is acetylated} \end{aligned} \tag{3.11}$$

Again, a complete position-based distribution for each modifications could be added as its own scoring module, but n-terminal acetylation is one of the most important position-specific modifications. The factor of 2 scaling is equivalent to the belief that if there is one or more acetylation, but none are on the n-terminus, this is a m̈istake$^{\prime}$. A good compromise would be to split the mod x aa into three different modules, one each for n-terminal aa, c-terminal aa, and internal aa. But this data was not readily available, so the simple rule R_2 was used.

3.3 Using PIE to Analyze Data

Together the nine scoring modules allow PIE to consider a wide variety of data and find the optimal modifications that account for that data.

Using PIE to analyze a data set and determine what modification variants of a protein the data requires four steps: setup, convergence, profiling, and interpretation. During setup, data to be integrated by the PIE are collected and formatted, and a parameter file controlling how PIE runs is configured. During convergence,

multiple small runs are performed using differing search path lengths. These are used to estimate convergence rates and to find the highest scoring answer. During the profile phase, a large run-set is generated at a run-length that results in convergence only some of the time. This profile is a distribution of best and nearly best answers and provides information such as whether one or multiple isoforms may be present, how good the data was in total, and how valuable each data set was individually. Interpreting this profile is the final step.

3.3.1 Setup

Setting up a PIE run involves collecting and editing data files needed as input to pie as well as setting `key` = `VALUE` parameters in the configuration file. The PIE has a modular design, allowing it to integrate multiple data types, depending on what scoring modules are used (e.g. Intact-mass, peptide MS/MS, prior knowledge, etc). The configuration file tells PIE what scoring modules to use and where to find the data, but it also contains sections to provide basic molecular data, parameters to the underlying MCMC statistical engine, and a section specifying what results PIE should generate and how to report them.

What is know about a protein and its modifications determines the modules that will be used to incorporate that data into a scoring landscape. If good data are available–data that provide a complete and consistent picture of a protein and its modifications–the PIE will be able to find unique high scoring answers for the modification state of the protein. If insufficient data are available, or the data sets contain convoluted or contradictory information, the PIE will likely still provide useful knowledge by characterizing the modification scenarios that are supported, to what degree they are supported, and how each data type individually contributes to that support.

Scoring modules

The scoring modules are each referenced by name in the parameters file, and have a boolean parameter that turns them on or off. If a module is turned on, then all parameters needed by that scoring module must also be provided. Details are provided in appendix C, PIE User Manual, but the scoring modules available are described above. First, however, PIE needs to know a little chemistry.

Molecule data

The PIE needs to know the sequence of the protein whose modifications are being studied as well as the masses for amino acids, adduct modifications, and small molecules like water. The mass files are table-based file, each row in these files represents a different molecule, and each column describes some basic property of that molecule such as a (globally) unique name, aliases, and the average and monoisotopic masses.

The adduct modification file provides mass data for the functional groups that may bond with a protein creating PTMs. The data in this file was taken from Proclame (Holmes 2004), but is also available from sites like `http://www.unimod.org/` (Creasy 2004). In addition to the mass of the modification, the mass value accounts for any molecular gains or losses during binding. For example in a typical methyl group addition, the methyl itself is 16.04 Da, however, both the protein and the methyl must loose a hydrogen (1.01 Da) to allow covalent bond formation, so the net addition is $16.04 - 2.02 = 14.02$ Da. Adducts with multiple binding modes may need to be listed multiple times.

The list of adduct modifications is also used to define the set of modifications the PIE will search for, and so may need to be edited to add modifications of interest, or

remove extraneous ones. The PIE uses modification-specific priors, which describe the general likelihood of seeing each modification type, and these need to be provided for new modifications as descried in the prior data section below. The number and masses of the modifications selected have consequences for the accuracy and running time for the PIE. The discussion section addresses specifying a modification set in more detail.

The PIE reads protein sequence information from the `targets.fasta` file. Unlike most other data files which are table based, this file is in standard FASTA file format (`http://www.ncbi.nlm.nih.gov/blast/fasta`). PIE identifies a protein by name from the definition line, the text up to the first space or the end of the line, whichever comes first. Data for multiple protein sequences can be specified at once in a single FASTA file, but only one protein at a time can be used by PIE for PTM identification in the current version. Each of the experimental data files contains a column that references a protein name, binding each row to a specific protein. The protein name must be identical (case sensitive) across all uses. This is a planned extension point, and in future it is possible that every data module will allow specifying the protein name it uses separately.

Experimental data

Experimental data is the main source of information used by the PIE to select and localize modifications on proteins. Pie currently supports three types of experimental data: an accurate intact mass, a set of peptides matched to the target protein via a program like GFS (Wisz 2004) or MASCOT (Perkins 1999), and a set of sequenced peptides or fragments with exact modification positioning information such as can be provided by MS/MS experiments (Searle 2005). Each experimental data type has its own scoring module and data file.

Intact mass data obtained through high-resolution mass spectrometry such as from an FTICR or orbitrap instrument is provided to the program through a data file and evaluated using the `IntactMassScoring` data module. This provides the total experimental mass for the modified protein being analyzed and the approximate mass error (in ppm or absolute mass units). Mass accuracy must be greater than 1/2 the mass of the lightest modification to prevent spurious modifications from being suggested due to error alone. However, better accuracy is generally needed. Low accuracy results do not provide sufficient constraints on total masses to limit combinatoric answers, and require complete peptide coverage to offset. The more accurate the intact mass, the more resolving power PIE will have when deciding between nearly equivalent answers. For example, if greater than 0.03 Da accuracy is available, PIE will be able to distinguish between trimethylation and acetylation adducts. Only one intact mass measurement is used by the PIE for any analysis, but multiple measurements can be present in one file as long as each has a different protein target name.

The `FragmentScoring` data module and its data file processes information from MS or MS/MS peptides that have been matched to the target protein via a program like GFS (Wisz 2004) or MASCOT (Perkins 1999),. This module is designed to handle peptides where only the presence/absence of a modification is known, but not its specific location within the peptide. Such data might be produced by the matching of peptide precursor masses (before fragmentation) to predicted peptides from the sequence. While less informative than data with a localized modification, these are nonetheless useful to the PIE, particularly for excluding PTMs from regions of the protein where matching peptides indicate no modifications are present.

The data file contains columns that specify the protein a peptide matched to, its alignment to the protein, and any (unlocalized) modifications associated with

the peptide. The scoring module tries to handle peptides that overlap and present contradictory information about what modifications are present. It also assumes that only partial coverage of the protein is available. Where peptides are available, PIE will use them to guide choice and placement of modifications implied in top-down data. Section 5.2.2, Evaluating a peptide data set, has more details..

The `LocalizedFragmentScoring` data module is designed to incorporate information from peptides or fragments identified from MS/MS spectra where a specific modification site is identified (or excluded). This module uses an input file containing columns specifying the name of the protein, the sequence provided, where that sequence aligns to the protein and the modifications present and their specific aa locations. This is very useful information, as it directly localizes a specific modification to a specific amino acid, reducing the size of the puzzle PIE must solve.

Priors data

Experimental data is the best way to determine what modifications are present and where they go on a protein, but real-world data are often incomplete and sometimes contradictory. When evaluating and interpreting less than perfect data, an expert relies on prior knowledge and experience. The PIE uses prior data models to accomplish a similar task. These models use input data such as the relative probabilities of modifications taken over all proteins described in a resource like Uniprot (`http://www.uniprot.org/`). Three prior models are available to provide information about the expected distribution of adduct modifications, two prior models describes cleavages, and one allows specific rule-based biases to be applied.

The two cleavage models are actually implemented by one `cleavageScoring` module and is based on a simple open and continue model similar to sequence alignment affine gap scoring (Gotoh 1982) with four terms (one pair for each end of

the protein) as parameters. No text file is needed. These parameters are provided directly in the modifications file.

The three modification distribution models consist of the `modCountScoring` module that applies a distribution of the expected number of modifications as well as the `modTypeScoring` and the `modLocationScoring` modules which each use text files based on database scanning to predict and localize modifications.

The `modCountScoring` module uses a simple center and error model. As with the `cleavageScoring` module, no separate data file is needed, parameters are provided in the `run.properties` file. If error value is small, then only modification scenarios with about the number of modifications expected (the center) will be predicted. If the error is large, then the center value is used as a guide, but easily ignored (See 5.2.4, Setting the number of modifications).

The `modTypeScoring` module uses a data file which contains a non-normalized weighting of how often each possible modification is expected. The `modLocation-Scoring` module uses a data file which is a table of weights, giving for each possible modification and amino acid combination how often that specific modified amino acid is expected. In order to allow for novel modifications, an additional (pseudo-count) of 1 is added to all possible modification-amino acid pairings, even those never previously found. To prevent any chance that a given pairing will be suggested, a count value of -1 can be used. The default data in these files The default values for the two database derived data files are taken from dbPTM (Lee 2006). However, representing as it does data averaged over all proteins, the defaults will need to be modified in many instances based on information about what is likely in the usually restricted domains that apply to specific experiments (See 5.2.3, Using domain specific prior information).

The rule based module, `ruleScoring` will be expanded in future to allow for

more convenient configuration, but currently allows setting parameters for two specific conditions: How likely n-terminal acetylation is, and how likely amidations and deamidations occurring in the same candidate are. The purpose of this data module is to include odd bits of prior belief that might apply in a given situation.

Template files are available in the distribution for all prior scoring functions needing separate data files. Appendix D (Tutorial Walk Through) describes how to copy and modify these files as needed for specific examples.

3.3.2 Setting up the parameter file

All runtime parameters for the PIE are provided through a configuration file, specified in a `key = VALUE` format. This configuration file is divided into three main sections: the first is the scoring section with parameters needed to configure the data and data models, a second section provides parameters to the underlying McMC statistical engine, and the final section specifies what results to generate and how to report them. Adjusting these parameters is required not just to tailor PIE for specific analyses, but also to run PIE, as this file takes the place of command-line options and parameters.

Data and data model parameters

The `Data and Data Model Parameters` section is subdivided into parts, one for each scoring module available. These select and set the values for the scoring modules described above, as well as identify the data files to read. Additionally, parameters are set in the section to define the default directories in which the PIE will look for its data files, including the basic molecular data files needed by PIE and the information needed to describe the protein target.

Pie must be able to find the data it needs. This is done by specifying up to three directories. Data files are loaded first from the `defaultDataDir`, then from the `experimentSetDataDir`, then from the `experimentDataDir`. Any file found in more than one directory will be loaded only from the *last* directory it is found in. This is useful when investigating how modification scenarios change when most of the data remains the same. The simplest configuration–used by default–is to read all data from one directory (`defaultDataDir`), leaving the other directories unspecified.

The names for each of the molecule information files and for the FASTA file containing the target protein must be specified. Since files can contain data for multiple proteins, the target protein name is needed as well.

McMC parameters

The defaults for the MCMC section do not generally need modifying and can remained unchanged except for special fine-tuning.

Run and reporting parameters

The last section of the properties file contains the parameters used to tell the PIE how to run and output results. A single run of the PIE usually consists of multiple searches, controlled by setting a `runCount` value. During each search, the number of steps used to find the best answer is controlled by a `maxSteps` parameter. Together these two parameters are the main controls for running the PIE. Other than the `startSeed` parameter, all other parameters in the section affect reporting only.

The `startSeed` parameter should usually be left at its default setting. Non-zero values cause the PIE to behave deterministically and can be useful for testing.

Setting this specifies a fixed sequence of pseudo-random numbers to use as a source of randomness by the PIE. The sequence used for any given run of PIE is available in the log file PIE generates as it runs, and if `startSeed` is set to value from the log file, results will be identical.

3.3.3 PIE's output

The PIE generates two result files–one summary and one detailed–and a log file. By default all results are generated into a directory created at runtime under the output directory specified. The created directory will automatically be named after the date and time the PIE was run, such as `2009_10_12__21_55_00_032`. This prevents accidentally writing over previous data. It is often a good idea to rename this directory to something more meaningful.

Both the summary and detailed result files are tab-delimited text files, with a header line identifying the contents of each column. The log file is simply a narrative of what the PIE does as it happens. Names of the result files can be changed through the obvious parameters, but since new output directories are create every time the PIE runs, it is ok to leave the defaults.

The level of detail in the log file can be controlled by changing the `logFilterLevel`. The default setting is generally adequate, recording an outline of what the PIE does including copies of all non-data messages. However, setting the level from `INFO` to `DEBUG` or `DEBUG_LOW` will provide additional levels of detail in the log file.

The summary file will have a number of lines matching the `runCount` parameter, one entry for each search. Each entry describes the best candidate found during a search, the modification pattern most consistent with the data. The information reported about each guess includes the the step on which the highest scoring answer was found, its score, and the modifications predicted. It also contain a separate col-

umn for the score generated by each of the individual component scores, allowing for detailed interpretation of results (see Section 3.6, Interpreting the Answer Profile).

The detail file reports the current state of the McMC search at given intervals throughout the run, each time the search has taken the number of steps specified by `everyN`. This output will be repeated for every one of the replicates specified by `runCount`. Reported information is similar to the summary file, except it is only the current state of the walk, not the best state so far.

The current state of the run will be echoed to the console at given intervals throughout the run. This includes a state description each time the search has taken the number of steps as specified by `consoleUpdate`. along with a messages at the start and end of each replicate giving the time.

3.4 Determining the Convergence Length

Simulated annealing by MCMC is a stochastic search method. It finds global optima based on a guided tour through a large set of candidate answers, but how long it must walk around before being guided to the best scoring candidate is unknown at the start. Quitting too early results in a less-then-best candidate. The average search length needed to find the best candidate is the convergence length, and this varies with the details of the scoring methods and data that makes up the landscape walked over. Deciding how long to walk for is necessary, but calculating this directly is theoretically difficult or impossible. Making things harder is the fact that more than just the single, best scoring, optimum candidate is needed; the nearly-best, sub-optimal candidates are needed too.

To obtain the needed convergence length an empirical approach based on a guess and refine strategy is used. The recipe for determining convergence requires using

PIE in an manner similar to guessing a number given feedback that a guess is too small or too large. Starting with an initial guess for a convergence length and then running repeatedly, each time using the results of the latest run to modify the convergence length on the next run allows determining the convergence rate. This can be repeated to obtain any degree of accuracy, but to minimize computation the target rate of convergence is 20% in a run of at least 10 samples (i.e. at least 2 with the same highest scoring value). Since this can happen by chance, a confirmation run of at least 10 samples at a longer run length with 3 or more of the same highest scoring value is used to provided supporting evidence that the real highest scoring answer has been found. Since every run starts from a random place, repeatedly finding the same maximum answer is taken as evidence that the true best answer has been found. If a new, higher scoring answer turns up, then estimation starts over with the convergence length that produced the new guess.

This same procedure can be used to optimize other parameters–such as how big to make each step. Although an exhaustive analysis has not been performed, most of these other parameters seem to have relatively little effect, or change little from run to run. The length of walk is by far the most important. Only after the convergence length has been determined can a profile of candidate answers be efficiently generated.

3.5 Obtaining a Complete PTM Answer Profile

Once the 20% convergence length estimate has been determined, it is easy to generate an answer profile to get an overview of the possibilities indicated by the data. To do this, a large sample of answers is taken with PIE, 100 or more searches. Since the chosen run length resulted in 2 of 10 results having the top-scoring value, about

20 of the profile searches are expected to result in the same high-scoring answer. The other 80 or so represent a sampling of near-optimal answers. This distribution over near-optimal answers provides a great deal of information about the way the data is structured and what kind of answer the best answer is. Sub-optimal answers that score high relative to the best answer indicate answers with less but significant support given the available data landscape. This includes answers that might represent alternate isoforms or answers that, due to missing data or data with insufficient resolving power, are indistinguishable.

3.6 Interpreting the Answer Profile

A step-by-step profile interpretation is included as Appendix D, Tutorial Walk Through. Only a brief summary of the profile is presented here, covering three separate ways to extract useful data from the profile.

A simple interpretation of the profile involves comparing the best and second best scoring examples. If the ratio of the two scores is far from 1, only the single best answer is highly supported. If the ratio is around 2:1, there is significant difference in support between the two answers, but some support for other answers exists. This is due to the built in "error at 1/2" scaling of each scoring module.

After the profile is sorted by total score and visualized, additional information is provided by the consistency with which different modifications are predicted across multiple guesses. If, for example, the same set of modifications and cleavages are used for most of the high scoring answers, varying only in the location of adducts, this is a good sign that these are the modifications present. Likewise, if a modification position is conserved through multiple guesses, that indicates strong support for that feature in the data.

Since each module's score is reported in addition to the total score, looking at the ratio of the component scores provides information as to which data types are providing the most discriminating power between the two best answers. For instance, if this is the peptide model, PIE is indicating that the second best scoring answer disagrees with one or more peptides. If it is the modification location prior, PIE is indicating that main difference between the two answers is that the better guess puts modifications on AA that are less common on average. The entire profile can also be sorted and visualized based on any of the component scores, showing how the guesses as a whole are viewed in light of each data type separately. This ability to investigate why PIE picked one answer over another is a valuable feature and provides significant insight into the story the data is telling about a protein.

By providing a way to look beyond the best answer and examine how different data types support and answer, and how each modification is supported provides a great deal of information about the protein isoform or isoforms represented by the data.

Chapter 4

Results

4.1 Validation Using L7/L12 Theoretical Data

PIE was first tested using synthetic data, based on experimental results from a survey of E. coli ribosomal proteins (Kevin Rankisoom - unpublished data). Manual interpretation of experimental data on the L7/L12 protein (Kevin Rankisoom - unpublished data) suggested the presence of several isoforms, one of which had three modifications: an N-terminal methionine loss, an N-terminal acetylation (on 2S), and a lysine methylation of (82K). Using a theoretical lysC enzyme digest from PeptideCutter (Gasteiger et al., 2005) a complete, ideal set of experimental data matching this isoform was generated, consisting of all bottom-up peptides, complete tandem MS/MS sequence data, and exact top-down mass. Several data sets with various levels of completeness and error were then produced from this ideal set by removing peptides and MS/MS sequence, and by adding error into the intact mass. An estimate for the intact mass accuracy is also needed; errors were chosen that are near or possibly larger than the imposed error to show how PIE performs with less than ideal data. Predictions obtained by PIE are presented in Table 4, "*Analysis of L7/L12 Experimental Data*".

To correctly characterize modification isoforms from typical proteomic experiments requires obtaining enough data to determine what modifications are present and where they are located. For this target isoform, just two peptides (with sequence) serves to identify the location of the acetyl and methyl adducts, and a moderately accurate intact mass–within 0.5 Da of the actual value–provides evidence that the only other modification is a loss of methionine. At the targets intact mass of 12,220 Da, 0.5 Da is about 40 ppm. Any greater intact mass error would support the addition of an amidation or deamidation modification ($\pm$ 1 Da).

The program converged to the correct answer with a few minutes of run-time for all theoretical L7/L12 data sets where there was enough data to localize the modifications (sets 1, 2, 3). By using prior scoring modules, PIE was able to obtain consistent answers even when either the intact mass (7 and 8) or the peptide data (4,5,6,9, and 10) did not contain enough information, i.e. when the intact mass error was large, or when MS/MS data or peptides are missing. This includes leaving out all peptide and MS/MS data for one or more modified peptides. In general, the lack of experimental localization information leads to multiple equal scoring answers different only in the position of modifications, but prior scoring modules help to order subsets by probability and rule out many unlikely answers (i.e. a phosphorylated arginine), and in some cases obtain the correct localizations (i.e. an n-terminal acetylation). For the remaining two data sets (11 and 12) the error-adjusted intact mass was off by more than 40 ppm and no peptide data was used. Here the answers obtained by the PIE unsurprisingly do not match those expected for this target isoform, but instead are more consistent with the given data; calculated intac masses for the guesses and within 10 ppm of the bad intact mass provided as experimental data to PIE.

Each theoretical L7/L12 data set was profiled to characterize the quality of the

proposed answer. The top two answer and the ratio of their scores are given in Table 2, "*Analysis of L7/L12 Theoretical Data*". Score ratios are consistent with the ability of PIE to provide greater discrimination between answers when more data is provided. The highest ratios are obtained for data sets 1, 2, and 3, which are the only ones containing all the minimum required information for complete characterization of the isoform. For each of these, the second best answer scores lower due to its contradiction of experimental data, as indicated in the why column of the data table. This column also shows that, for the remaining data sets where not enough experimental information is available, PIE is using prior expectations to select the best answer, but this is accompanied by lower score ratios.

4.2 Analysis of H23C Theoretical Data

As a test of the PIEs ability to handle more complicated modifications patterns, including conflicting peptide data, a synthetic data set based on two theoretical isoforms of the human h23c histone protein was generated. This protein was chosen because biologically it is highly modified and presents a much more complex target than L7/L12. Considering two different isoforms simultaneously allows testing how PIE handles conflicting data. One virtual isoform, H5, was imagined with 2 methylations, 2 acetylations, 1 phosphorylation, and an N-term met loss. The other virtual isoform, H7, was given two additional phosphorylations (See Fig. 4, "*H23C Theoretical Peptide Data*"). From these two virtual isoforms, four artificial data scenarios were created: one data set containing peptides, tandem sequence, and intact mass consistent with H5, one set consistent with the additional phosphorylations present in H7, and the remaining two sets having combined bottom-up data consistent with a mix of the two isoforms, but using either the H5 or the H7 intact

mass. This data and PIEs predictions are presented in Table 3, "*Analysis of H23C Theoretical Data*".

The program was run assuming each intact mass was in errors by +10 pp m when measured by an imaginary instrument giving ± 20 ppm measurement error. MS/MS and peptide data (including all modifications) was assumed to be present covering 75% of the protein. For the combined peptide data sets, two regions of the protein have both a correct and an incorrect peptide assigned to them, the incorrect peptides originating from the other isoform. The PIE was able to correctly identify all modifications and their positions for the H5 and H7 pure isoform data and for the H5 mixed case; for the H7 mixed case it correctly identified all modifications, misplacing just one of the phosphorylations with conflicting data. For the H5 mixed case, this means PIE correctly identified and localized all modifications, and was able to ignore peptide data indicating 2 additional phosphorylations that did not apply to H5 isoform. For the H7 mixed case, this means PIE is reaching the limit of its understanding of the data. To correctly place this modification, the PIE needs additional information or modified scoring: See Chapter 5, Discussion, for more information.

4.3 Analysis of L7/L12 Ribosomal Extracts

After validating the PIE on theoretical data, it was applied to data from the L7/L12 ribosomal extracts collected during an investigation of the role and extent of ribosomal PTMs in E. coli K-12 (Kevin Rankisoom - unpublished data). L7/L12 is particularly complicated and was chosen because there are multiple isoforms simultaneously present in the sample, testing the PIEs ability to handle heterogeneity. Results are presented in Table 4, "*Analysis of L7/L12 Experimental Data*".

Top-down (intact mass) measurements were collected from several ribosomal extracts analyzed on two different mass spectrometers: a Bruker BioTOF II time-of-flight MS and a Fourier transform ion cyclotron resonance (FTICR) MS. Mass resolution for the BioTOF typically runs around 20 ppm and for the FTICR around 1 ppm or better.

A total of nine intact masses were selected from the MS data as corresponding to isoforms of L7/L12. The intact scoring model requires some estimate for the accuracy of these masses, although a precise estimate is not needed. The expected accuracy for the analyzing instruments is a much simpler and easier estimate, and would also work, but the presence of internal standard analogs provides the opportunity for a second estimate. By calculating the mass error for all other apparently unmodified ribosomal proteins identified in the extract, a better estimates is likely obtained. Misidentification of one or more protein as unmodified is possible as these are not true internal standards, but this only makes the error estimate more conservative. The intact error windows used in Table 4, "*Analysis of L7/L12 Experimental Data*" are those that would contain most data points, excluding outliers.

Corresponding bottom-up peptide data for L7/L12 were obtained from E. coli K-12 ribosomal extracts by digestion with trypsin and analysis on a QSTAR MS/MS Quadrupole time-of-flight. Eighteen unique peptides were identified by precursor masses including six with adduct modifications. MS/MS sequence was obtained for six peptides, including three of those with modifications (see Fig. 5, "*E. Coli L7/L12 Peptide Data*").

The PIE was applied to each of the nine intact mass targets, scoring the combined intact mass and bottom-up experimental data along with all available prior data models using default parameters. Fig. 6, "*Convergence and Profile Sampling, L7/L12 220-1H*" show two of the run sets used to identify convergence for one of

the targets, as well as an answer profile. The PIE converged to a best prediction for each of the nine targets, describing three different isoforms. The results, presented in Table 4, "*Analysis of L7/L12 Experimental Data*", were consistent with a manual interpretation for eight of the nine intact targets. Three separate isoforms were identified, 12,206, 12,175 and 12,220. These were each consistent with prior manual analysis, including localization of one to three modifications and an n-terminal methionine cleavage. One of the nine intact masses, putatively representing a fourth isoform at 12,163 Da, was incorrectly predicted to be mono-methylated; manual interpretation suggests this isoform is unmodified except for n-terminal methionine loss.

Each of the nine targets was profiled along with the second best guess and the relative score ratio for these top two guesses are included in the data table. The answer profile in Fig. 6, "*Convergence and Profile Sampling, L7/L12 220-1H*" shows the top 100 results for the 12,220-H1 isofrom. For this trimethylated prediction, there is not enough data to localize two of the methyl adduct, so there are many nearby answers differentiated only by placement of the modifications. When the nearby best answers have similar scores, the score ratio is near 1, indicating multiple answers are supported by the data.

Chapter 5

Discussion

5.1 Discussion of Results

To investigate the feasibility of McMC/SA based data integration with PIE, I sought answers to questions such as: Could PIE avoid being lost in endless sea of equal-mass answers, particularly given only incomplete data? Can PIE handle data sets that have dependencies, errors or contradictions? Can it find answers in a reasonable time? Can PIE be implemented using a modular approach so it can readily accommodate additional data types in the future?

Data integration in proteomics is difficult because data is usually incomplete and the solution space is large, especially when proteins have multiple modifications. Given the results obtained with PIE, McMC and simulated annealing appears to be a useful way to approach this problem. Other methods are possible, and predictions for simply modified proteins, where data represents a single protein isoform with only one or two modifications, is easy. For instance, an exhaustive exploration of a constrained space, such as is done by PROCLAME (Holmes and Giddings, 2004) in searching for top-down candidates could be extended to include simple peptide or sequence data. However, the situation rapidly becomes complex with more than just

a couple of modifications. The combinatoric explosion of possible answers require algorithms–like simulated annealing–that scale well. These are also the cases where the most interpretive help is needed.

As originally envisioned, PIE was not intended to handle bottom-up data representing multiple isoforms, but during testing and validation, it became apparent that an intact mass alone (guided by simple priors) contained enough experimental information to identify a likely set of modifications (as in Table 2, "*Analysis of L7/L12 Theoretical Data*", data set 10). This suggested that bottom-up data with peptides representing several isoforms simultaneously could be used, relying on the intact mass to distinguishing the relevant peptides. PIE did surprisingly well interpreting these more complex mixtures, even coming up with an answer not previously considered during manual analysis.

Having multiple interpretations for the same data set makes mass spectrometer data difficult to analyze. For example, the data for the L7/L12 isoform 12,206 supports either trimethylation or acetylation. Which is correct? Given the available mass spectrometer data, it is impossible for human or software to tell; both isoforms may be present. PIE's strong support for only the trimethylation prediction results from the multiple methylated peptides in the bottom-up data. It is interesting that prior to using the PIE I had not previously considered trimethylation since manual interpretation with an acetylation was so obvious. Unfortunately, without assuming completeness, there is no reason to believe all the intact masses have been found, and hence no reason to believe the dimethylated peptides (Fig. 5, "*E. Coli L7/L12 Peptide Data*") have to belong to any of the intact mass isoforms. The only solution to these issues is to acquire more data, but how and what data? This is essentially a resource availability question, and depends on the specifics of the experiment and data. If, for example, ultra-high mass accuracy instrumentation was available (i.e.

an FTICR-MS) it would be possible to resolve the ~ 0.03 Da difference between a trimethylation and an acetylation.

The lack of specificity in mass spectrometry data makes it critical for automated analysis software to integrate a wide range of data. Without the flexibility to incorporate new techniques, any such software will quickly become obsolete, whereas software supporting a wide variety of data allows the experimenter to choose the fastest, cheapest, and most accurate methods to produce data. Modularity is central to the PIE, allowing simultaneous integration of a variety of both MS and non-MS based data. New modules can be added with relative ease, and it is easy to run what-if experiments including or excluding data.

One aspect of the way PIE uses bottom-up data is to specify what not to look for. Modifications not detected in bottom-up data are less likely to be present. PIE uses this negative information to apply Occam's Razor and favor simpler answers. For example, without peptide data, the intact data model causes PIE to add extra amidations or deamidations ($\pm$ 1 Da modifications) to better match any deviation in intact mass greater than 0.5 Da, even if such deviation is due to measurement error and not the presence of a modification (as in Table 2, "*Analysis of L7/L12 Theoretical Data*", data set 12). Where bottom-up data was available and no such modifications were seen, the chance that such a modification would be proposed was reduced.

5.1.1 Model and data accuracy

Scoring is affected both by the accuracy and completeness of the available data as well as the accuracy of the model. Prior modules are based on information obtained from databases that suffer from ascertainment bias. The modifications present in the PDB, for example, are not independently sampled from all proteins, but are

more like an applause meter, where more popular or interesting proteins and their modifications are over-represented. Simple priors allow for fast calculation; given the bias in the underlying data, additional effort to refine them to provide highly accurate prior models seems at this time to be unproductive.

The intact mass model is dependent entirely on the accuracy of the measurement. As shown in the results (i.e Table 2, "*Analysis of L7/L12 Theoretical Data*", Sets 11 and 12), if the intact mass has enough error, PIE will find a consistent answer that is also in error. Although PIE performed surprisingly well even with wide mass tolerances, narrower error windows increase discriminating power.

The peptide model had difficulty dealing with isoform mixtures due to data conflicts inherent in a bottom-up shotgun approach. Several models were tried but no simple model allowed discrimination of multiple isoforms to my satisfaction. The current mass-based model suffers from over-weighting peptides with the most variants, allowing this to override other information. Compared to the consistent trimethylated proposal obtained for the L7/L12 12,175 isoform, the methylation proposed for the 12,163 variant is not supported by the intact mass. Here matching to the intact mass data by the intact mass model is outweighed by stronger matching to the peptide data as evaluated by the peptide model. This is due to the large number of methylated peptides, and might be avoided with an improved scoring model. Increasing accuracy in intact mass would also eventually reverse this, producing the expected manual answer. Additional data on the relative abundance of peptides could help identify the most prevalent isoform, but would decrease the ability to identify all others. The underlying McMC and SA algorithms can optimize continuous values as well as discrete ones, and it is possible that PIE could be extended to include guesses with quantification estimates for an isoform.

PIE uses a score ratio derived from the answer profile in lieu of a formal error

model (e.g. there are no p-values). The answer profile samples directly from the empirical distribution, with the ratio of any two scores giving the ratio of their probabilities. It also represents how unique a given answer is. Given that the incomplete and ambiguous nature of MS data supports multiple similar answers, it is important to determine if other good answers are likely. It is not clear how to generate a more meaningful error model. Bootstrapping can have difficulty with extreme values (Kysely, 2009) and a pure McMC sampling (without the simulated annealing optimizations) is computationally expensive. Additionally, these or similar error models only provide probabilities or confidence intervals with respect to the model used. The modular data framework used in the PIE is designed to allow users to change models easily.

5.2 Using the PIE to Solve More Complex Problems

The analyses presented are relatively straight forward applications of the PIE. Many situations arise when doing proteomic analysis that do not look like these examples. A few thoughts on possible techniques for using PIE in more complicated situations are provided here.

What if the sample analyzed consists of a mixture of different proteins, not just different isoforms? A separate run for each possible protein can be done. Although scores are not comparable between runs with different data or protein targets, the number of replicates of the highest scoring value and the ease with which convergence is obtained can be used as a rough guide to the best targets, and bad choices for proteins will likely result in the uninformative prior (unmodified protein).

What if there are many isoforms, or a large number of identical modifications

varying only by modification position? Nothing will help distinguish positions other than position-specific data. This could be ms/ms data indicating amino acid specific modification information, or a more accurate prior distribution, such as the variable weighting of phosphorylation sites generated by Netphos (Blom et al., 1999).

What if the protein has long truncations due to signal peptide removal? Using a data module that scores cleavages based on information from a program like SignalP (Bendtsen et al., 2004) would improve results. It is also easy to run such a program separately and try the predicted truncated protein as a target.

What if the protein contains point mutations? These can be considered modifications, although care must be used when cyclical modifications that result in a net 0 mass are possible, i.e if. K $\Rightarrow$ T, T $\Rightarrow$ S, and S $\Rightarrow$ K are all allowed, many isobaric modification sets are possible (with $1, 2, 3, \ldots, N$ sets of these three modifications).

What if a list of adduct modification names for the mass deltas is not available? If bottom up data contains an identified peptide but with an un-identified mass shift, this might be entered in the `modification.txt` file as an unknown adduct ..., with appropriate mass data. However, it is hard to provide the requisite prior information for this modification relative to other modifications or target AAs.

What if adduct modifications are variable, such as polysaccharides or lipids? If you can not easily list each individual specific mass for all the possible modifications, the PIE can not currently predict these modifications. It might be possible to extended PIE such that it could allow for variable modifications, such as by adding or subtracting pieces of a modification instead of an entire modification at once.

What if there is no intact mass data? Intact mass data is an critical piece of information. Without this, modifications will be lost in the gaps, the regions of the protein not covered by peptide data, as there then no experimental data describing them. It may be possible to run the PIE multiple times varying the numbers of

expected modifications parameter of the modCount data module. See 5.2.4, Setting the number of modifications.

What if there is neither useful top down nor bottom up data? There is no experimental evidence, and the PIE has nothing to work with. The PIE can only integrate the data you give it. Without experimental evidence, only the uninformative prior (unmodified protein) result is obtained.

5.2.1 Specifying the target modification set

The modification set considered by the PIE for a given run is provided in the modifications.txt file. Not including a modification in the modification set is one of the few actions that can cause correct answers to be excluded from the search space considered by the PIE. To allow for novel solutions, it is best to leave the list as long as possible, with two caveats. First, if the list gets too long, run-times will be extended, and second, the answer space may become too diffuse. Even though the PIE does not suffer from exponential explosion in computing time, every modification specified does increases the number of steps needed to converge, and hence increases the average running time. Adding a reasonable number of modifications should not adversely affect performance; adding all the modifications listed in dbPTM (Lee 2006) likely would.

Currently, adding a modification to the `modifications.txt` file also requires adding associated data to the files used by the `ModType` and `ModLocation` prior data modules as described in the Materials section.

5.2.2 Evaluating a peptide data set

MS/MS scoring data - usually but not necessarily from bottom-up style experiments - provide a rich but mixed source of information about the location and type of a protein's PTMs. The PIE performs best when the MS/MS information is consistent, but can handle inconstancies and errors in the data. However, there is no one obvious way to interpret fragment data, and several scoring algorithms have been tried. Different methodologies are currently implemented though the choice of different scoring models, each with their own benefits and drawbacks (see C.2.5, Scoring Module Descriptions, for more details). Currently the two best scoring strategies are `deltaMass` and `errorCounting`, with the `deltaMass` model used by default.

The `deltaMass` strategy is a mass-based evaluation model, similar to that used for the intact mass, but covering only a fragment of the protein. This can be used when only precursor masses or peptide mass fingerprint style data are present.

The `errorCounting` strategy looks for individual mis-matches within the modifications allowed for the aligned fragment. This does not use the mass of the fragment, only its interpreted sequence and modification components from a program that can localize the PTM (e.g. Findmod, (Wilkins 1999)).

5.2.3 Using domain-specific prior information

One of the design goals for the PIE is to allow easy incorporation of various types of knowledge about a problem. Sometimes this knowledge is in the form of experience about what the answers should and should not look like within the range of experiments a user is familiar with. The PIE makes use of such information through explicitly stated prior distribution of expected results. For example, the distri-

butions of expected modification types and locations is likely different in general between histones, ribosomal proteins, membrane proteins, or signal transduction proteins. Specific data sets can be acquired to accurately compute weightings in specific circumstances, and then turned into a separate scoring module. One module planned will apply a distribution for phosphorylation of eukaryotic proteins based on the predictions of the program NetPhos (Blom 1999). This could be done for other PTM prediction program. Although not a substitute for good experimental data, this might allow the PIE to fill in gaps where MS data are missing.

Manually setting prior weightings allows the PIE to take advantage of domain knowledge based bias even in the absence of complete or accurate data. This has the added value of ensuring that any such bias is applied explicitly and consistently - if phosphorylation is impossible, it can be removed from the list of modifications. To ensure an n-terminal acetylation occurs, its weighting in the `ruleScoring` module could be changed to a very large number.

The PIE could also be used to evaluate what if data scenarios, such as slowly and repeatedly decreasing the likelihood of phosphorylation and running the PIE to see how the predicted candidates change.

5.2.4 Setting the number of modifications

In the absence of other data, one reasonable prior for the number of modifications would be to set the expected number of modifications to zero, and the allowed range of modification numbers to the length of the protein. This guides selection of the fewest modifications that are consistent with remaining data.

Without experimental intact mass data, integration of results becomes much harder, and the PIE is forced to rely on this prior as the guide for predicting modifications in regions of the protein not covered by peptide or fragment data.

However, it is still possible to explore the space of candidates consistent with the peptide or fragment data by using a sequence of values for the number of expected modifications and setting the allowed range of modifications very low (like 0.25). The results from a sequence of the PIE runs, each at a different expected number of modification, then summarize the supported isoforms assuming one modification, then assuming two modifications, etc.

5.2.5 The need for an accurate convergence parameter

Long searches result in finding the best scoring answer almost all the time, and hence provide high confidence that the correct answer has been discovered. However, large overestimates for the needed walk length is not acceptable for generating answer profiles. Besides the waste in computational effort, the landscape of the solution space is never revealed and competing solutions are never seen. It may be the case that there is only one best answer worth considering. But it may also be the case that there are many scenarios of very similar scores, with close to equal support given available data. If there are alternative modification scenarios that are almost equally supported by a data set, these would not then be found if the run length is too long. Also a problem is when if the data is too noisy, incomplete, or conflicting to isolate one or more isoform with the best support, it is possible that one answer is slightly elevated in score over others, and hence get reported as the best answer. A measure of the quality of this answer is needed, and is provided by the "almost-but-not-quite-best" answers as revealed by the profile obtained using an appropriate search length.

5.2.6 Identical candidates vs. identical scoring candidates

To be clear, PIE is concerned with the score of guesses, not the actual guess itself. Where modifications are not localized by available data, it is common to have several different but identical scoring candidates. This is usually readily apparent in the profile view. For example, if a protein had three equally valid serines that could be phosphorylated and there are no evidence for which of the three is modified, all of these are scored as equal best answers. All three different answers will be present in the result, each with a different phosphorylated serines. When determining convergence, this is unimportant. The score is the same for each and so this would count as three replicates of the same best score and hence evidence for convergence.

5.2.7 The uninformative prior Result

It is possible that there may not be enough good data for PIE to find any meaningful result. If experimental data is particularly uninformative, a prior only result may be obtained. An answer based on the prior data instead of the experimental data is uninformative because the same answer will always result, generally an unmodified protein. For such results, convergence is usually obtained slowly, though this may depend on the data provided. One way to check this is to run the PIE without any experimental data, then compare the result to one with the experimental data provided. If the answers are similar, this indicates that the experimental data are not adding to the result, and hence are for some reason uninformative. The key is then to determine what is missing so that additional experimental data can be provided to provide a more informative answer.

5.2.8 Supporting R scripts

Several R-based analysis scripts are provided in the PIE distribution in the `pie/R` directory. These can be used with the R statistical analysis software, available from The R Foundation for Statistical Computing (`http://www.R-project.org`). The profile graphs, Fig. 6, "*Convergence and Profile Sampling, L7/L12 220-1H*" and Fig. 9, "*L16 Answer Profile*", were generated with these scripts.

5.3 Conclusion

The current version of the PIE is only the first step in creating a tool that can integrate MS data and predict PTMs, but already it shows great promise. Using simulated annealing allows the PIE to explore the unfathomably large solution space of all possible modifications of a protein, and find the consistent answers. It is surprisingly robust, capable of decomposing an intact mass into a likely combination of modifications, and with the addition of MS/MS data, even a complex mixture of overlapping and conflicting peptides from several isoforms can be used to obtain specific modifications localizations. The PIE provides a integrated approach for combining top-down and bottom-up MS data in the context of prior knowledge to automatically determine the PTMs associated with a protein. By starting with few assumptions about the answer needed and using a flexible, modular framework that lets the data provide the constraints, the PIE can be extended and improved as new or better data and data models become available.

Appendix A

FAQ

A.1 What is PIE?

PIE is short for the Protein Inference Engine. It is designed to predict posttranslational modifications by integrating a variety of data, but most significantly by combining mass spectrometer data from top-down (intact mass) and bottom up (peptide and MS/MS) experiments. It does this using a guess and refine methodology based on Markov chain Monte Carlo and Simulated Annealing. Although the methodology itself is very interesting process, it is not necessary to understand how PIE works to use it. All thats needed is to select which data evaluation modules you want PIE to use, format your data to be read by these modules, and set up one or more configuration files.

A.2 What will PIE tell me?

After several steps, PIE will provide you with the best prediction it can find given your data. This answer is a set of PTMs and their locations, including both adduct modifications like phosphorylation or methylation as wells as n-terminal and c-terminal truncations. It will also provide a list of nearly-best answers that, while not the best, are still pretty good. Answers are ranked and have a relative quality score, which helps determine how good the best answer is and whether it is unique

or if the data supports multiple answers. Additionally, PIE provides a detailed explanation for its ranking and scoring decisions in the form of sub-scores for each data type evaluated. These sub-scores help identify problem data and allow for additional validation or interpretation of answers.

A.3 What kind of data can PIE use?

PIE can potentially use any kind of data describing the modifications of a protein, but each type of data must have an associated scoring module that allows PIE to evaluate and use the data. The most useful information is experimental data about a specific protein being investigated and its PTMs, but PIE also uses background data about proteins in general or for the specific domain of proteins being studied. Data is usually input through a table-based text file or as a couple of parameters in the configuration file. Each scoring module is specific to one type of data and, when provided with a set of modifications (and their locations) on a protein, returns a score that represents how well the data supports those modifications. To add a new scoring module for a new data type requires adding or modifying a java class to hold the new type of data, and adding a class that implements the scoring interface as it applies to the new data type. Currently, PIE has support for the following data types (See appendix C, PIE User Manual, for additional information):

Experimental MS data types

Intact mass: The total mass for a protein and its modifications.

Peptide mass: A list of masses for (potentially modified) peptides and how they align to a protein.

Peptide sequence: A list of sequences aligned to a protein. If the sequence covers a modified region of the protein, the modifications must be identified and localized.

Background or prior data types

Adduct abundance: Some adducts are common, others are rare.

Adduct location: Every adduct has its own preferential AAs.

Adduct count: How many adducts are present, on average.

Cleavage length: How likely is 0, 1, or longer n-terminal and c-terminal cleavages.

Explicit data rules

N-terminal acetylation rule: Acetylation is often n-terminal, regardless of the AA present at that terminus. How often this happens is important information and helps with providing accurate results.

Amidation-deamidation pair rule: Amidation and demaidation are not often found on the same protein, and pose special problems for PIE if they are allowed. How often this happens is important both for accuracy and for efficiency.

A.4 I have some data about PTMs. Can PIE help me?

That depends on what data you have, and what you want to know. There is fairly a long list of caveats and exceptions that follow, but PIE currently requires some

effort to use, and this list is intended to help you avoid wasting time. If you are not sure, go ahead and try to use PIE. Even if today PIE can't help you, it is designed for easy extension in multiple ways, and almost all the limitations below can theoretically be overcome, at least partially. We welcome comments on what PIE could or could not do for you.

A.4.1 Do you have MS data?

If not, PIE may not be useful to you. It can combine the result from multiple prediction tools, and if that is your intent, then great. But without experimental data, it will difficult for you to evaluate if a result is biologically meaningful.

A.4.2 Is the sequence of the underlying protein known?

PIE requires that you break your data up into queries about 1 protein target at a time, where the (canonical) sequence of the protein is known. If you are only looking at results from a purified protein, great. If you have multiple protein targets, each must be run separately, and on any given run of PIE you must tell it the one sequence you are targeting on that run and give it the data that applies to that one protein sequence only. PIE *can* handle data representing several different modification isoforms, but only a limited number at once.

A.4.3 Is your peptide data targeted?

PIE requires that only a limit number of peptides not matching the isoform queried be present. If you have peptide or MS/MS data collected from a small pool of proteins, PIE may be able to work with that, but will evaluate the data in terms of

only one intact mass at a time. The more dilute the peptides from that target are, the worse PIE will perform. PIE is intended to work with peptides pre-identified as associated with a protein by some other software, such as GFS (Wisz et al, 2004) or MASCOT (Perkins et al., 1999).

A.4.4 Can you specify a discrete list of all modifications?

All modifications that are possible must be in this list, and each may not span amino acids. This means methylation and dimethylation both have to be specified, and that modifications that span AAs (like cystine crosslinks) need special rules to ensure they occur in sets. It also means that variable modifications like lipids and sugars can not be analyzed for.

The shorter the list of modifications simultaneously searched for, the better. Both performance and accuracy degrade as the number of potential adduct modifications increase. In addition, each modification simultaneously searched for must have all needed prior data specified to use an associated prior scoring distribution (e.g. relative abundance or amino-acid preference).

A.4.5 Does the protein analyzed have any sequence mutations?

The canonical sequence required by PIE must match that of the protein being analyzed. N and C terminal deletions are handled separately, and are not considered mutations. Insertions and deletions are generally not allowed, except possibly as specific point mutations that can be considered as adducts.

Single specific point mutations (e.g. S $\Rightarrow$ T) can be considered as an adduct with an appropriate delta mass, $massof(T) - massof(S)$. Point insertions can be

considered likewise as S ⇒ ST, and point deletions as S ⇒ null. This is somewhat unsatisfactory, and problematic as any combination with an adduct also needs to be included separately, (S ⇒ T+phos, etc.). PIE has only been tested searching for about 10 adducts. Including all 18 non-0 mass AA mutations X 10 possible adduct modifications even for just one mutable AA transition adds 180 more to the adduct list; this is currently too large a list to allow PIE to work efficiently.

A.4.6 Do you have enough data to determine an answer?

If you don't have enough data to constrain the possible solutions and actually determine an answer, PIE can't help. But you don't really need to answer this, PIE makes the best of the data it has, and will produce blurry guesses even with weak data. PIE will let you know how poor your data is by not finding answers, by reporting the "no answer" answer of "unmodified", or by finding many indistinguishable answers. If you can add more data, you can rerun PIE to refine answers.

A.4.7 Congratulations!

If you made it this far, then PIE can help you to quickly and automatically analyze your data. It will provide the most-likely modification pattern represented by your data, including AA localized adducts, N-terminal cleavages, and C-terminal cleavages. It will also provide a description of the ensemble of nearly-best answers, allowing you to understand in a deep way just what it is that your data is telling you about what modifications are present, which parts of the prediction are more or less sound than others, and the relative contribution of each data type.

A.5 I want to use PIE! What do I do, specifically?

Perhaps the best way, if you have never used PIE before, is to run one on the demo projects distributed with PIE. See Appendix D, Tutorial Walk Through.

A.6 Now that I have run PIE, what do the answers mean?

PIE provides both a best answer and a range of nearby answers to aid in the interpretation of results. Appendix D, Tutorial Walk Through provides step-bystep instructions on how to generate some results, and then describes how to interpretation them.

A.7 If PIE didn't provide good answers, what do I do now?

PIE should have given you and indication as to why it could not find a good answer using your data. This is one of its strengths, being able to provide reasons for its answers. Perhaps you can do some manual data cleaning to help PIE focus on the key elements, or perhaps you can collect more or better data. See section 5.2, Applying PIE to More Complex Problems, for more details.

Appendix B

Installing and Running PIE

Version 0.3 of the PIE has been tested for installation and execution (with Java 5) mainly on Apple OS X based systems, but it does run under Microsoft Windows as well. The present version runs only from the command line.

B.1 Pre-requisites for PIE

The PIE has the following pre-requisites:

- Java 1.5 or greater (`http://www.java.com/en/download/manual.jsp`).
- A user with some familiarity with the command line and with the basic concepts of mass spectrometry-based proteomics.
- A text editor to format input information.
- Some way to view and manipulate the results (e.g. R or Excel).
- That you agree to the non-commercial license PIE is offered under.

B.2 The PIE Distribution

The PIE can be downloaded from `http://pie.giddingslab.org`. It is distributed as a compressed file. Download the latest version (highest numbered) file and

unpack it. The resulting folder/directory, named after the version (something like `pie-0.5.1`), will contain the PIE application files (in the `./bin` subdirectory), template input files (`./data`), one or more sample runs including input data and results (`./demo`), documentation (`./doc`), and some sample R scripts (`./R`).

B.3 Installing and Running the PIE Directly from the Jar File

The PIE is written in Java and packaged in a jar (Java Archive) file, `pie.jar`, located in the `./bin` directory of the distribution. It is easy to run this on any system at the command line using the generic `java -jar` command with two (or more) arguments: the `pie.jar` file, which is the application, and one or more property files containing all the necessary run parameters. These files include pointers to the data files that the PIE will read and integrate. For example:

```
> java -jar /path/to/pie.jar /path/to/run.properties
```

The `run.properties` file can be obtained by modifying the template from the distribution's data directories, or from one of the demo directories. Further details of the PIE's operation are provided in the user manual (Appendix C).

B.4 Installing the PIE as a Unix Command-line Application

Although not necessary to run the PIE, it is easy to install the PIE on a unix-based operating system so that it can be run without needing to use `java -jar` for every invocation. The `pie.sh` program in the `./bin` directory of the distribution is provided for this purpose. It is a simple shell file that wraps the java call and

uses an environmental variable to find PIE, and this allows PIE to be installed as any other application on a unix system. Note, this file will not work in a Microsoft Windows based environment.

One possible way to install the PIE as an application (assuming you have sys admin access), is given below.

1. Move the unzipped PIE distribution folder somewhere useful, for example to `/usr/local` and then in the `/usr/bin` directory, create a link (not an OS X shortcut, those are different!) to `bin/pie.sh` within the distribution folder in the bin, named something like `pie` to allow easy updating to new versions. Due to the protections on these directories, you may need to authenticate:

```
> sudo mv pie-0.1.1 /usr/local/
> cd /usr/local
> sudo ln -s /usr/local/pie-0.1.1 /usr/local/pie
```

2. Create a link named PIE to the executable file `/usr/local/pie/bin/pie.sh` in `/usr/bin`.:

```
> sudo ln -s /usr/local/pie/bin/ /usr/bin/pie
```

3. Add an environmental variable `PIE_HOME` that points to where you put the PIE directory link. It is simplest to put the following line in your `~/.profile` file (create this file if it does not exist):

```
export PIE_HOME=/usr/local/pie
```

4. To pick up this new environmental variable, you need to log out and back in again, or source the profile directly:

```
> source ~./profile
```

PIE can also be installed to run as `pie` for all users on a machine by adding the `PIE_HOME` variable to the machines root profile, and can be called from other programs if the `PIE_HOME` variable is set in the `.bash_rc` file for the user or in the machine level variant.

If there are problems using `PIE_HOME`, the `pie.sh` file can be edited to look for the PIE link wherever it lives.

You can now run PIE from the command line simply by using the command `pie` and passing it the name of one or more configuration files.

```
> pie /path/to/run1.properties /path/to/run2.properties
```

Appendix C

PIE User Manual

This user manual describes version 0.3.5 of PIE.

C.1 Overview

This manual contains four sections. This first section outlines the organization of the User Manual and describes the basics of how data gets into and out of the PIE. The remaining four sections describe the molecular data files used to teach the PIE chemistry, the properties file that controls how the PIE does everything, the scoring modules used to evaluate data, and the output files that contain the PIE's predictions regarding the modification patterns it has inferred from the data. For detailed tutorial walk through, see Appendix D.

PIE requires one or more configuration files, which take the place of complicated command-line option sets. The configuration files are implemented as Java properties files, but are used simply to provide a set of `key=value` parameters to PIE. Details are given in section C.3 on the configuration file, including a description of what most parameters do. Some parameters provide the names and locations of the data files to read in and the result files to write out. Other parameters provide information related to scoring functions, and these are described with the specific scoring modules in section C.4 to allow for a modular documentation style.

Most data files read by PIE are tab-delimited text files with rows of data organized into columns, the first line giving the column names. Some data files are associated with specific scoring modules. C.4, the section describing the scoring module, provides

information describing these file as well as the associated properties. Other files are core data files used by PIE to load chemical data such as amino acid names and masses or protein sequence. Section C.2, gives the specifics for these core data files.

PIE's main output is a summary file containing the best scoring candidate from a run set. The highest score representing the best guess (or guesses) found. There is also a detail file that contains a sampling of guesses taken along the way as PIE searched for each candidate. This is useful to see how PIE is thinking and to diagnose what went wrong. A log file is also produced the generates a narrative of what PIE is doing It contains some generically useful information as well, including the random seed used and the time PIE is taking to find each candidate, which can be used to predict how long a run-set will take to finish. Section C.5 describes the output files in detail.

The parameter and data files are read once when PIE starts up, and then are not used again. However output files (summary, detail, and the log file) are each updated repeatedly as PIE runs, so moving or locking an output files will cause PIE to fail.

Tab-delimited text file format

Almost all files read by PIE are tab-delimited text files with the first row of data interpreted as case-sensitive column names. These files work like a data tables or spreadsheets. Each type of tab-delimited file will have a different set of required columns. Column order is unimportant, extra columns (with unique headers) are ignored. Any row starting with a # symbol is considered a comment line and is also ignored. Comments are valuable as they allow a block of descriptive text to start files, and allow individual rows to be commented out, easy switching data rows on or off and preserving the context for rows that remain.

The header row must be the first non-blank, non-comment line in the file. If any required column name is missing or repeated, PIE will exit with an error. Depending on the file, PIE will also expect certain things from the data elements in a column, such as

being non-blank, or being a floating-point number. Failure to understand or read a data element will also cause PIE to exit with an error. Details on what columns are required or optional and on what is expected of data elements in each column is given with the description of the associated file. Tabs are adjusted to make examples display nicely.

Specifying the filenames of data files

All data files are specified by `key` = `fileName` entries in the run.properties file. Data files are always identified with a property key, and hence can be named anything. If `filename` is a relative path, including just a file name, PIE will look sequentially through up to three different base directories (the `defaultDataDir`, `experimentSetDataDir`, and `experimentDataDir`) and then try the current run directory. The first file matching the given filename will be used. If no such file is found, PIE will exit with an error. See section C.3, PIE Configuration File Reference, for more information.

C.2 Core PIE Data Files Reference

C.2.1 Overview

Four core data files are always required by PIE regardless of the scoring modules used: `proteinFastaFile`, `aaDataFile`, `molDataFile`, and `modDataFile`. The `proteinFastaFile` file must always be specified, but a default version of the remaining three molecular mass files are provided by default if not explicitly set. This section describes the four core files.

C.2.2 The proteinFastaFile

PIE needs to know the name and sequence of the target protein. This must be provided in a standard FASTA file identified by the filename property `proteinFastaFile`. Which one of the (possibly many) sequences in that file is specified via the `targetProteinName` property. This name is case sensitive. The first whitespace-terminated block of text on each definition line is checked for a match, starting from the top of the file. The sequence for the first protein with a matching definition line is read.

Parameters

proteinFastaFile = "`file.fasta`" The name of file providing protein sequence information to PIE.

targetProteinName = "`aProteinName`" The name of protein sequence to read in from the fasta file, case-sensitive and may contain no white space. For example:

`proteinFastaFile` = "`This`"

would match a protein with definition lines like

>This is a protein

> This is a protein

> This

but would *not* match

>this is a protein

>. This

>This.

>This1

Columns

This file has no columns, it does not use the tab-delimited file format of most PIE data files, but instead is a standard FASTA file. FASTA files are ubiquitous, but a reference for the basic format is `http://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml//`. The sequence following the selected definition line is read and will be used as the canonical target protein sequence by PIE. Only the base 20 AAs single character representations are recognized, case insensitively. Blanks and numbers within the sequence are ignored.

Example

An acceptable FASTA file specifying the sequence for three proteins, "L7/L12", "L16", and "h23c", might be:

```
>L7/L12
1  MSITKDQIIE AVAAMSVMDV VELISAMEEK FGVSAAAAVA VAAGPVEAAE EKTEFDVILK AAGANKVAVI
71 KAVRGATGLG LKEAKDLVES APAALKEGVS KDDAEALKKA LEEAGAEVEV K
>L16 gi|16131192|ref|NP_417772.1| 50S ribosomal subunit protein L16 [E. coli K12]
MLQPKRTKFR KMHKGRNRGL AQGTDVSFGS FGLKAVGRGR LTARQIEAAR RAMTRAVKRQ GKIWIRVFPD
KPITEKPLAV RMGKGKGNVE YWVALIQPGK VLYEMDGVPE ELAREAFKLA AAKLPIKTTF VTKTVM
>h32c
MARTKQTARKSTGGKAPRKQLATKAARKSAPATGGVKKPHRYRPGTVALREIRRYQKSTELLIRKLPFQRLVRE
IAQDFKTDLRFQSSAVMALQEASEAYLVGLFEDTNLCAIHAKRVTIMPKDIQLARRIRGERA
```

C.2.3 The aaDataFile

PIE needs to know amino acid masses to calculate the theoretical weight of proteins. This file provides the names and abbreviations as well as residue masses for the 20 standard amino acids. Similar files are used to provide data on modification adducts and other molecules. The values for the data in the default `aaDataFile` were obtained from `http://proteome.gs.washington.edu/cgi-bin/aa_calc.pl`.

Usually the default data file is fine and this does not need to be specified. If most of the instances of an amino acid "X" are expected to be modified (e.g. it is a cystine derivative, or isotopically labeled), it might be more efficient to specify a different default mass for the affected amino acid in this file, and add the rare unmodified case, "unmodified-X", to the `modDataFile`.

Parameters

aaDataFile = "`aa.txt`"

Columns

Code: The one-letter code for an AA, a unique row id.

Abbreviation: The three-letter code for an AA, a unique row id.

Name: The full name for an AA, a unique row id.

MassAvg: The average mass of the AA residue.

MassMono: The monoisotopic mass of the AA residue.

MassMai: *Unused* The most-abundant isotopic mass for an AA residue.

Example

```
Code Abbreviation Name           MassAvg  MassMono  MassMai
A    Ala          Alanine        71.07880 71.03711  71
R    Arg          Arginine       156.1876 156.10111 156
N    Asn          Asparagine     114.1039 114.04293 114
D    Asp          Aspartic acid  115.0886 115.02694 115
...
```

C.2.4 The molDataFile

PIE needs to know the mass for water to calculate the theoretical weight of proteins. This file provides the name, abbreviation, and masses for water. Similar files are used to provide data on modification adducts and AA residues. The values for the data in the default file were obtained from `http://www.unimod.org/`.

The mass of water is added to the summed residue mass to obtain the unmodified protein mass for theoretical calculation. (H for the 5' n-terminus, OH for the 3' c-terminus). Almost certainly the default `molDataFile` is fine and will not need to be specified. Although not supported at the current time, this file is also the place where other non-protein molecules might be defined, such as bound ligands, etc.

Parameters

molDataFile = "`mol.txt`"

Columns

Abbreviation: A short code for the molecule, a unique row id.

Name: The full name for the molecule, a unique row id.

MassAvg: The average mass of the molecule.

MassMono: The monoisotopic mass of the molecule.

MassMai: *Unused* The most-abundant isotopic mass for the molecule.

Example

```
Abbreviation  Name   MassAvg  MassMono    MassMai
H2O           water  18.0153  18.010565   18
```

C.2.5 The modDataFile

PIE needs to know which modification adducts to look for and the mass changes they induced when added to amino acids. Pie uses the amino acid residue mass plus the adduct delta mass given here to calculate the theoretical mass of a modified AA residue in a protein. Similar files are used to provide data on amino acid residues and other molecules. The values for the data in the default file were obtained from `http://www.unimod.org/`.

As the modifications specified in this file are the only ones PIE will find, the more modifications in this file, the more likely PIE will find the correct answer for your data. However, the more modifications in this file, the longer (possibly much longer) PIE will take to determine the answer. Each modification also requires significant additional information to use basic prior scoring modules. It is up to the user to determine the optimum list, as modifications common in one context may not be common in another.

The current implementation allows only a fixed delta mass for each specified adduct, and requires a one to one correlation between adduct and AA. Variable modification adducts such as sugars or lipids can only be searched for if a specific delta or small set of specific deltas can be defined. Compound adducts like dimethylation must be listed as a single entry, separate from monomethylation or trimethylation. For adducts that span AA, the modification must be broken and each piece listed separately. This is simple only for symmetrical adducts like cross-linked cystines. Unfortunately neither proximity nor pairs (complete sets of adduct pieces) is checked for.

Parameters

modDataFile = "`mod.txt`"

Columns

Name: The unique full name for a modification.

Abbreviation: A unique short multi-letter code for a modification.

Code: A unique, case sensitive one letter code for a modifcation.

MassAvg: The change in the average mass due to the modification.

MassMono: The change in the monoisotopic mass due to the modification.

MassMai: *Unused* The change in the most-abundant isotopic mass due to the modification.

Example

```
Name            Abbreviation Code DeltaAvg  DeltaMono DeltaMA
Acetylation     Acet         A    42.0367  42.010565  42.0
Amidation       Amid         I    -0.9848  -0.984016  -1.0
Deamidation     Deam         i    0.9848   0.984016   1.0
#Farnesylation  Farn         N    204.3511 204.187801 204.0
Methylation     Meth         M    14.0266  14.015650  14.0
Phosphorylation Phos         P    79.9799  79.966331  80.0
```

C.3 PIE Configuration File Reference

C.3.1 Overview

The PIE requires a significant amount of effort to set-up and run. This is unavoidable given the goal of integrating a wide variety of different kinds of data. Bulky data is read from spreadsheet-like, tab delimited text files, but simple data along with all control and configuration data is read from a configuration file. This file conforms to the Java properties file specification (`http://download.oracle.com/javase/6/docs/api/java/util/Properties.html`), and is given as an argument when running PIE. It contains a number of `key = value` lines, each defining a property. The key is constant and used by PIE, but the value associated with that key is set in this file and loaded at runtime, allowing easy configuration of data and control information. The Java properties specification allows for comments; anywhere a # character appears, it and the rest of the line is considered a comment and is ignored.

Using multiple configuration files

It is possible to use more than one file of properties when running the PIE; indeed this is the recommended procedure. All files are combined in the order specified into a single collection of properties. Using more than one file is useful for maintaining different sets of defaults that vary in purpose or granularity. For example, it is possible to define one file containing system level default properties for PIE, another containing default properties for a given set of experiments, and a third containing only those properties changing for a specific run of PIE The command line looks like:

```
> java -jar pie.jar base.properties exp.properties \
    run.properties
```

The properties from each file are stacked one on top of the other. Where the same key is present in more than one file, only the value from the *latest* file is kept. Usually files

are listed from the most global to the most local. Multiple configuration files provide a flexible and efficient way to organize input, reducing the amount of setup effort required by reusing previously configured information.

The minimal configuration file

To further reduce the complexity of setting up a run, most properties have reasonable default values specified, but some *must* be specified. They are given the default value "`OVER_RIDE`", which can be used in any configuration files as a signal that the associated key must have a value specified by a configuration file loaded after it. The keys that must be provided under all circumstances are:

`targetProteinName` = "`OVER_RIDE`"

maxSteps = "`OVER_RIDE`"

runCount = "`OVER_RIDE`"

These parameters are described more fully in their respective sections, below.

Note: The configuration file keys and their default settings are one part of PIE where significant changes are to be expected. A "What's New" document included with future releases of PIE will describe new, deprecated, or dropped keys, changes in how a key's value is interpreted, changes to default values, and any change to the set of required keys

C.3.2 Configuration file properties

Organization of the configuration file

The configuration file is separated in to three sections. The first, `Data and data models` is concerned with input and configuring the scoring information. The second, `McMC Configuration`, gives tuning parameters for the McMC and SA methods used to search for answers. The third section, `Run and Reporting`, specifies runtime monitoring and output. Generally, changes will be made infrequently if at all to the `McMC Configuration`

section while changes to the other two sections are needed with different experiment sets. Some of the `Run and Reporting` parameters will change every run.

Filename parameters

Where parameters provide filenames, these can usually be absolute or relative. Any relative filename, including just a plane filename with no path, will be searched for across a specified set of directories starting with the current working directory.

C.3.3 Configuration file section: Data and data models

This section defines the directories PIE will search, the scoring modules PIE will use, and the external data files that PIE will read. Each scoring module has a property named like `is‘`*`scorer`*`’scoring` that enables the module if set `true` and disables it if set `false`. For scoring modules that requires external data there is also a property named `‘`*`scorer`*`’DataFile`. The currently available experimental scorers are: `intact`, `fragment`, and `localizedFragment`, the prior-type scorers are `modType`, `modLocation`, `modCount`, `cleavage`, and a special scoring module, `rule`.

Block: Data directories

The search path PIE uses to find data is composed of the local directory PIE was run in, followed sequentially by the next three directories. PIE will stop searching with the *first* file found. Default values are as shown, corresponding to the current directory (again), the parent directory, and the parent's parent directory.

`defaultDataDir` = “`./`”

The first directory on the data path, after the local directory. By default this is also set to the local directory.

`experimentSetDataDir` = “`../`”

The second directory on the data search path. By default this is the parent of the current directory.

`experimentDataDir` = “`../../`”

The second directory on the data search path. By default this is the parent of the parent of the current directory, two levels up.

Block: Molecular data files

Pie requires the name and masses of various molecules and molecular pieces. These are specified by four external data files.

`molDataFile` = "`molecules.txt`"

The file describing small molecules and their masses, including water.

`aaDataFile` = "`aminoacids.txt`"

The file describing amino acids and their masses.

`modDataFile` = "`modifications.txt`"

The file describing the modifications to search for, including the mass changes they cause.

`proteinFastaFile` = "`target.fasta`"

A standard `FASTA` file giving protein sequences.

`targetProteinName` = "`OVER_RIDE`"

The name of the protein (from the `FASTA` file) to use as a target.

The `proteinFastaFile` is the source of target protein sequence, as specified by the `targetProteinName` and the `modDataFile` is also used as the list of modifications to search for. For details on these data files see C.2, Core PIE Data Files Reference.

Block: Experimental data and scoring parameters

This block provides experimental data to PIE, configuring the three experimental data scoring modules. Each of the three scoring modules–`intact`, `localizedFragment`, and `fragment`, requires an external data file. The `fragment` scorer also takes several additional parameters. If not using a particular type of data, the `is`*`'scorer'`*`scoring` parameter should be set to `false`, and all parameters associated with that scorer are then ignored. If a data type is evaluated, all associated parameters are required and will be validated,

especially filenames. If a file can not be found on the data path and is not provided by default to PIE, it will cause and error. More details about the experimental scoring modules are provided in the following section.

`intact` scorer (molecular mass data)

`isIntactScoring` = `false`

Use intact mass data? If true, reads the `topDownDataFile`.

`topDownDataFile` = "`intact.txt`"

The name of the data file containing the intact mass data.

`fragmentLocation` scorer (peptide sequence data)

`isLocalizedFragmentScoring` = `false`

Use peptide sequence data? If true, reads the `localizedFragmentDataFile`.

`localizedFragmentDataFile` = "`localizedFragment.txt`"

The name of the data file containing the peptide sequence data.

`fragment` scorer (peptide mass data)

`isFragmentScoring` = `false`

Use peptide mass data? If true, reads the `fragmentDataFile` and the following parameters.

`fragmentDataFile` = "`fragment.txt`"

The name of the data file containing the peptide mass data.

`fragmentScoringAlgorithm` = "`deltaProductMass`"

The method used to score the provided data. The other option is `errorCounting`.

`fragmentMassType` = "`AVG`"

The method used for measuring the mass. May be "AVG" for average mass, or"MONO" for monoisotopic mass. "MAIM" for most abundant isotopic mass is not yet implemented.

`noveltyFactor` = `0.5`

How much to penalize guesses that include modifications *not* supported by any detected fragments. Default is to be conservative with guesses. setting to 0.9 makes PIE more open to picking novel modifications. See C.4.8, The `modTypeScoring` Module, for additional details.

Block: Prior data and scoring parameters

This block configures the scoring models based on prior type information. External data files are required by the `modLocation` and `modType` scoring modules. The `cleavage`, `modCount` and `rule` scoring modules require no external file, all data for these modules are specified here as properties. As with the experimental scoring parameters, if a scoring module is not used, parameters are ignored. Additional details on how PIE uses data are provided in the section on scoring functions, C.4, below.

`modType` scorer (the relative frequency of modifications)

`isModTypeScoring` = `true`

Use data on how often a modification occurs? If true, reads the `modTypeDataFile`.

`modTypeDataFile` =“`modType.txt`” The name of the data file containing information about the relative frequency of different adduct modifications.

`modLocation` scorer (a modification's amino acid preference)

`isModLocationScoring` = `true`

Use data on the frequency distribution of a modification across the different amino acids? If true, reads the `modLocationDataFile`.

`modLocationDataFile` = “`modLocation.txt`”

The name of the data file containing information about thefrequency distribution of a modification across the different amino acids.

`cleavage` scorer (n-terminal and c-terminal truncation data)

`isCleavageScoring = true`

Use n-terminal and c-terminal cleavage? If true, reads the following parameters.

`cLoss = 0.8`

Penalty for loss of first AA from c-terminus.

`cLossMore = 0.8`

Additional penalty for eadd AA lost from the c-terminbal end, after the first.

`nLoss = 0.9`

Penalty for loss of first AA from n-terminus.

`nLossMore = 0.8`

Additional penalty for eadd AA lost from the n-terminbal end, after the first.

`modCount` scorer (how many modifications are expected)

`isModCountScoring = true`

Evaluate guesses based on the total number of modifications? If true, reads the following parameters.

`modRate = 0.0`

The expected number of modification on average, the mean, average, or center.

`modDelta = 10.0`

Expected spread in the number of modifications. 50 % of the time, the number of modifications should be within ± `modDelta` of the average `modRate`.

`rule` scorer (data for miscellaneous rules)

`isRuleScoring = true`

Use the various pattern rules? If true, reads the following parameter.

`ruleParameters` = "`amidDeam = 0.5, nTermAcet = 2.0, polyAmid = 0.5, polyDeam = 0.5`"

The value is itself a list of key=value pairs, separated by , (commas). Spaces are ignored. Values are all real numbers. Keys are the names of rules, only those shown above are allowed. See C.4, Scoring Module Reference, for additional details.

C.3.4 Configuration file section: McMC configuration

This section contains parameters that control the underlying McMC walk and SA optimization process. It should not generally be necessary to modify values here. The given parameters work well for proteins ∼100 AA in length that have relatively few modifications (cleavages or adducts).

Block: McMC proposal parameters

This block contains parameters that control the proposal distribution, determining where to step to next in the protein search space (See Fig. 3, "*Solution Space Representation*"). Relative to the current guess, the step could be: a `modificationChange`, with the new guess different by one randomly selected adduct modification at some random position, a `modificationSwap`, which exchanges the position of two randomly chosen modifications, a `cleaveNChange`, which adds or subtracts one amino acid (and any associated adduct modification) from the N terminus of the protein, and a `cleaveCChange` which does the same but from the C terminus. In the descriptions below, changes that result in the same guess are allowed, random means equally likely, and the total of all values must sum to 1.

The `nullModProb` is a parameter that affects how often, when doing a modification change, that the change should be to a null modification (unmodified), and how often it should be (each equally probably) one of the modifications listed in the `modDataFile`.

`modificationChange` = `0.5`

Pick an amino acid at random and change the modification on it to a null modification with probably `nullModProb`, or to one of the non-null modifications, each with probability $1/(1 - \texttt{nullModProb})$.

`modificationSwap` = `0.2`

Pick two modifications at random and swap the modifications on them.

`cleaveNChange` = `0.15`

Change the length of the modification by adding or subtracting one modification (equally likely) from the n-terminal end. If can't add, instead truncated down to 1 AA only; if can't truncate, instead fill the n-terminal end back to uncleaved.

`cleaveCChange` = `0.15`

As `cleaveNChange`, but adds/subtract from the c-terminal end instead.

`nullModProb` = `0.8`

How often, if changing a modification, that the modification should be "`null`".

Block: McMC walk parameters

This block controls the mode of operation for the sampler and associated properties. Scoring is ignored for `burnInSteps`, allowing a more random starting point for the McMC walk. If you have a protein with more than 100 AA or more than 10 modifications are searched for, this value should be increased proportionally. If `useAnnealing` is set `false`, PIE performs a pure Metropolis-McMCM (Metropolis et al., 1953) walk, although no Hastings correction (Hastings, 1973) is applied to counter the `nullModProb` induced bias towards a steady-state with the specified ratio of `null` (unmodified) amino acids.

By default PIE's stopping condition is a fixed number of steps (`maxSteps` *gt* 0, see below). If `acceptableScore` is *gt* 0, PIE will stop and report a guess as soon as its score equals or exceeds the specified score. If `maxNoImproveSteps` is *gt* 0, PIE will stop after a specified number of steps are taken without finding a better-scoring guess.

`burnInSteps` = `2000`

Number of randomized proposal steps to take without evaluation (always accepted).

`maxNoImproveSteps` = `0`

After this many steps without getting a better score, PIE will give up. The default, `0`, means never give up.

`acceptableScore` = `0`

Keep going until the specified score is equaled or exceeded. If too large, the score can never be reached and PIE could run forever (see `maxNoImproveSteps`. The default, `0` means pay no attention to the score.

`useAnnealing` = `true`

Use simulated annealing to find the best answer if true, otherwise do a (biased) McMC sampling.

C.3.5 Configuration file section: Run and reporting

This section controls how PIE outputs results while it runs and when complete. Several values have the default of "`OVER_RIDE`", indicating they must be set.

Block: McMC reporting parameters

This block controls final output location and format. The `outputDir` gives the directory to write to. It must exist or the current directory is used. If `isAutoOutputDir` is true (the default) then a subdirectory will be created off of the specified output directory, with a name corresponding to the time-stamp for the beginning of the PIE run. Three files are generated into the (time-stamp) output directory named as specified by `summaryResultsFile`, `detailedResultsFile`, and `logFile` parameters. The `summaryResultsFile` reports details on the best guess obtained for each of `runCount` runs (`runCount` is described in the next block, below). The `detailedResultsFile` gives results sampled at regular intervals (`everyN` steps, also descried below) throughout the search, for each run. The `logFile` provides details of what PIE is doing as it runs, including all input data, details of any errors, and the random seed used by PIE. The `logFilterLevel` can be used to control how much information is dumped to the log file. Details on this parameter and the contents of the output files are described in C.5, Output File Reference.

`outputDir` = "`./`"

The directory to produce output into, by default the current directory.

`isAutoOutputDir` = `true`

Generate a new directory in the output directory for each run of PIE, named for the date and time when PIE ran.

`detailedResultsFile` = "`pieDetails.txt`"

Name for the file where PIE outputs intermediate guesses proposed during each sampling walk of a run. All samples are appended at the end of each walk.

summaryResultsFile = "pieSummary.txt"

Name for the file where PIE outputs the final, best scoring guess for each sampling walk of a run. The best guess is appended at the end of each walk.

logFile = "pie.log"

Name for the log file where PIE will output errors, warnings, information, and debug information, as selected by the logFilterLevel

logFilterLevel = "DEBUG_LOW""

What information to output to the log. Allowed values are "ERROR", "WARNING", "INFO", "DEBUG", and "DEBUG_LOW". Also contains the random seed used.

Block: Main runtime parameters

Generally, PIE runs several searches that each walk for maxSteps steps. runCount is the number of separate sampling walks in a run. The best answer found for each walk will be output to the summaryResultsFile. During each walk, the current state of the walk will be output to the detailedResultsFile, everyN steps, and to the console every consoleUpdate steps (summaryResultsFile and detailedResultsFile are discussed above). The startSeed can be set to allow exact repeats of PIE, but by default is 0, implying a random seed should be automatically generated and used. The actual seed used can be obtained from the log file.

startSeed = 0

The random seed to use. If 0, generates a seed automatically.

startSeed = "50, -10, 64, 16, 23, 64, 80, -47"

An example of the format for the random seed, a comma-separated list of eight signed one-byte numbers. Spaces are not significant.

`everyN` = "`OVER_RIDE`"

Must be specified. How many steps to take between output of each intermediate result of a walk. This is repeated for each walk in a run.

`consoleUpdate` = "`OVER_RIDE`"

Must be specified. How many steps to take between each output to the console, providing feedback on the current state of a run. Some runs can take hours, so this is useful.

`maxSteps` = "`OVER_RIDE`"

Terminates the run after this many steps. Must be specified. If zero, better have set one of the other ways to end a run in the McMC Reporting Parameters block described above.

`runCount` = "`OVER_RIDE`"

How many sampling walks to take, the number of separate guesses to make in a single run.

C.4 Scoring Module Reference

C.4.1 Overview

To evaluate how well a guess matches all available data, the PIE uses a scoring function composed of multiple terms, where each term matches a guess to one kind of data. The terms are then combined (multiplied together) to get the total score. Each kind of data is treated independently, without regard for how well other terms match, and is implemented as a different scoring module. This modular scoring system allows PIE to easily adapt and include new kinds of data.

Each scoring module takes as input a specific kind of data D_k, and a guess g for the modified protein that, possibly, best represent the data. It returns a score S_k that represents how well the guess g matches the model for data set D. These scores are then multiplied together to produce the total score S for a guess over the total data D.

$$S(g, D) = \prod_k S_k(g, D_k). \tag{C.1}$$

The cost/benefit of assuming independence between scoring terms

The PIE treats each kind of data separately and independently. As with naive Bayes classifiers (Zhang, 2004), the modeling error this causes is a necessary trade-off to include a variety of data sources easily. The assumption is each data set contains significant information (orthogonal or independent relative to other sets), and so considering it increase the net ability to find answers. Even if there is some non-obvious significant dependency in one data set, it is only necessary to do well enough to get an answer.

Resource limits necessitate simple models

Each scoring module reads and interprets a set of data provided to PIE and returns, for any possible guess, a score representing how well that guess matched the data set. As such, each module represents a data model. providing an (un-normalized but finite) density function over the range of all possible guesses. This function has a maximum where the guess best matches the data, but also scores highly when guesses are nearly correct. The number of times this model must be evaluated (on the order of 10^6 to 10^8 times), imposes a strong practical constraint favoring simpler models. As with the assumption of independence, using multiple simple models allows the use of more data, and the combined information from multiple data sources is assumed to make up for information lost by simpler models.

Best vs. nearly best answers

One drawback of treating scoring independently is the difficulty in equating the meanind of the difference in scores (e.g. between the best and a nearly best answer) in different scoring models. This is addressed using a simple rule of thumb; each error should result in a difference factor of 0.5. One error means a 50% confidence window on continuously varying results, or a literal factor if counting discrete matches or mismatches.

Experimental vs. prior data

The PIE distinguishes between experimental data, which is data collected about the specific protein, and prior data, which are summary statistics about how proteins are modified in general, but may or may not apply well to the protein being studied.

Scoring modules descriptions

Currently PIE provides eight different scoring modules, three experimental and five prior. Additional scoring modules are can be easily created, although significant Java programming skill and some familiarity with the PIE program are necessary. A developers guide is planned that will contain information to make the process of creating new scoring modules easier.

All prior scoring modules are enabled by default, with reasonable default parameters. Two of the prior scoring modules are dependent on the selection of modifications to search for, but default scoring module data match with the default set of modifications to search for.

All experimental modules are all turned off by default as they require user-specific data, but the main use for PIE is to integrate experimental data, so usually one or more of these should be enabled and configured.

Each scoring module is described separately, with sections detailing the scoring model, configuration file properties, and the format of any associated data file. The scoring model section describes how the data is evaluated, gives a mathematical description of the model, and discusses how nearly best guesses are interpreted. For experimental models, how errors in the data effect results are also described; for prior models the equivalent information about what parameters do is provided. The configuration file section describes each parameter that effects the scoring module, giving the default value and describing interactions with other parameters. For scoring modules that need external data, a section describing the associated tab-delimited data file format will be provided, along with a sample file. The general format for data files is covered in the data file section (C.2).

C.4.2 The intactScoring module

A protein's intact mass contains a significant amount of information about its PTMs, wrapping up in one value information about what modifications are or are not present, as well as where cleavages have occurred. If the intact mass M_e of a modified isoform can be determined experimentally, it can be used to evaluate the accuracy of a guess. The greater the difference between the total theoretically calculated mass for a guess M_t, and the measured M_e, the worse the guess. The best guess will exactly match the experimental value, but any guess that matches within experimental error E is still a good guess. The scoring function used to model this is:

$$S = \frac{1}{|M_e - M_t| + E}$$

where

$$\begin{aligned} M_e &= \text{Experimental mass,} \\ M_t &= \text{Theoretical mass,} \\ E &= \text{Absolute experimental mass error.} \end{aligned} \tag{C.2}$$

The error E is often expressed relative to the experimental mass M_e, i.e. in ppm, but it is easily converted to an absolute mass error as needed for this formula:

$$E_{abs} = \frac{E_{ppm} \cdot M_e}{10^6} \tag{C.3}$$

To allow for a unified interpretation of error across all scoring modules, a strict interpretation of error would be the value giving the ±25% confidence widow for the measured mass, i.e. that the experimental intact mass is within $\pm error$ of the true value 50% of the time. Any reasonable estimate of the instrument error will allow PIE to use intact mass to pick the best answers, but makes the comparison with nearly best answers more qualitative in nature.

Parameters

`isIntactScoring` = `false`

Turns scoring by this module on (`true`) or off (`false`). By default intact scoring is turned off as it requires experiment-specific data.

`topDownDataFile` = "`intact.txt`"

If using `intactScoring`, PIE needs a data file. By default the file "`intact.txt`" will be searched for on the data path, but any name can be specified.

`proteinName` = "`OVER_RIDE`"

The target protein not only selects the `FASTA` file sequence to use, but also selects the line from the `topDownDataFile` that will be read and used as data for this scoring module during a run.

C.4.3 The intactScoring module data file

This file provides data from intact mass experiments. Each line represents the intact mass for a specific measurement or isoform, as identified by a unique Name (matched to the `proteinName` parameter. Only one intact mass experiment can be considered at a time, and PIE will use the row for the protein specified in the configuration file. PIE can use either average (`MassAvg`) or monoisotopic mass (`MassMono`) measurements, support for a most abundant isotopic mass measurement is not available at this time.

Columns

Name: The name of the protein isoform. May not contain any white-space, must be unique within this file, and one name must match the `proteinName` specified in the config file.

MassAvg: The experimentally determined average mass, or `NaN`. One, but only one, of `MassAvg`, `MassMono`, and `MassMai` must be specified as a integer or decimal number. The other two must be set to `NaN`.

MassMono: The experimentally determined monoisotopic mass, or `NaN`. See `MassAvg`, above.

MassMai: *Unused.* The experimentally determined most-abundant isotopic mass, or `NaN`. Unimplemented, currently should always be `NaN`.

Error: The instrument error, in either absolute (`ABS`) or relative `PPM` units. Must be an integer or decimal value greater than 0.

ErrorModel: Units used the `Error` specified above, either `ABS` or `PPM`.

Example

Name	MassAvg	MassMono	MassMai	Error	ErrorModel
L7-L12-A-ideal	12220.08	NaN	NaN	0.1	ABS
L7-L12-A-good	12219.80	NaN	NaN	20.0	PPM
L7-L12-A-min	12220.57	NaN	NaN	50.0	PPM
L7-L12-A-high	12221.00	NaN	NaN	100.0	PPM
L7-L12-A-low	12219.00	NaN	NaN	100.0	PPM

C.4.4 The fragmentScoring module

Bottom-up experiments can provide the masses of peptides aligned to a region of a target protein. Any adduct modifications are partially localized by the extent of the peptide, although not to specific AAs. By matching experimental peptide masses to the associated region of a guess, the quality of that guess can be evaluated. The more peptides that match and the closer the mass of a given peptide to the matching region, the better the guess. A perfect guess should match all peptides within experimental tolerances, although in some cases there will be contradictory or erroneous data, described in more detail below. Two different scoring functions can be used to model this, `deltaMassScoring`, which uses scoring similar to the intact scoring module, and `errorCounting`, which uses scoring similar to the localized fragment scoring module described in the following section.

PIE assumes all data it is presented with applies to the current prediction. It is possible for matched peptide lists to contain peptides from multiple isoforms of the target protein. The intact mass will help PIE select as a final answer one isoform compatible with both intact and fragment data (indeed this is the main driving force for creating PIE). However, if peptide data is lop-sided enough, data from this module can outweigh the intact isoform data, causing it to be ignored. If this happens the data set should be split with each set containing only one of the contradictory peptides, but all others. Running PIE separately on each of these data sets should then provide at least one answer consistent with the MS/MS and the intact mass. Note, if there are multiple points where contradictory peptides are present, then this may need to be done repeatedly.

It is also possible that peptides are present that match no intact mass isoform detected. Again, if enough of these peptides are present, they will distract PIE as it tries to match them. This may be useful as an alternate answer, but if the intact mass can be trusted to truly represent an isoform, another answer still exists. Again the solution is to split the data, splitting on conflicting data or possibly on data supporting one PTM, and trying again.

The deltaMassScoring algorithm

This algorithm bases the match scores for each peptide on the difference between masses of a peptide and its associated region of the guess, similar to scoring by the `intactScoring` module. A total score for all peptides is calculated by multiplying the individual peptide match scores together.

$$S = n^m \cdot \prod_i \frac{1}{(|M_{e,i} - M_{t,i}| + E)}$$

where

$$\begin{aligned} i &= \text{iterator over all matched peptides;} \\ M_{e,i} &= \text{Experimental mass of peptide i;} \\ M_t &= \text{Theoretical mass of guess region aligned to peptide i;} \\ E &= \text{Experimental error;} \\ n &= \text{novelty factor;} \\ m &= \text{count of novel modifications.} \end{aligned} \tag{C.4}$$

Modification novelty is explicitly part of the model. If a guess contains modifications that were not detected in bottom up experiments, the assumption is this is less likely to be correct. This implements a type of "Occam's razor" filtering, allowing PIE to preferentially select simpler explanations. Setting $n = 0.5$ is recommended to allow a simplified interpretation of an error, this is described below in the section on `errorCounting`.

Although the error E has a specific meaning (see the comment on error in C.4.2, The `intactScoring` Module), it is currently assumed to be 1, and cannot be user modified.

The errorCounting algorithm

This algorithm bases match scores on the number of matched AA and matched modifications identified by an external program such as GFS (Wisz et al., 2004) or MASCOT (Perkins et al., 1999). This is similar to the `localizedFragmentScoring` module.

$$S = n^m \cdot \prod_i \frac{1}{2^{a_i} \cdot 2^{m_i} + 1}$$

where

n = novelty factor,

m = count of novel modifications,

i = iterator over all matched peptides,

a_i = count of mismatched AA over aligned guess region for peptide i,

m_i = count of mismatched modifications over aligned guess region for peptide i,

$+1$ = small shift to avoid singularities.

(C.5)

By using a factor of two for each match, the ratio of any two guesses different by 1 error, either an unmatched AA or an unmatched modification, is 0.5. This helps maintain a quantitative interpretation nearly-best guesses relative to the best guess. The +1 only significantly effects values with very few mismatches, although this does shift the meaning of "error" somewhat.

See the previous description of the `deltaMassScoring` algorithm for details on the novelty factor.

Parameters

`isFragmentScoring` = `false`

Turns scoring by this module on (true) or off (false). By default the scoring of

peptides is turned off as it requires experiment-specific data.

`fragmentDataFile` = "`fragment.txt`"

If using `fragmentScoring`, PIE needs a data file. The file "`fragment.txt`" will be searched for by default, but any name can be specified.

`fragmentScoringAlgorithm` = "`deltaProductMass`"

Specifies which basic scoring algorithm is used. Two options are currently supported: "`deltaProductMass`", the default, which uses mass-difference based scoring and "`errorCounting`", which uses the putative sequence and modifications of an identified peptide along with an accuracy measure to compute the score.

`fragmentMassType` = "`AVG`"

Declares the type of mass value used in the associated data file. The default is "`AVG`" for average masses. "`MONO`" is also allowed, specifying that monoisotopic mass values are given.

`noveltyFactor` = `0.5`

Multiply the returned peptide score by this value for each modification that is of a type not detected in the peptide data. If used with the default `deltaProductMass` scoring algorithm, the modifications column of the associated data file will need to contain accurate data. To turn off novelty scoring, set this to 1.0.

`proteinName` = "`OVER_RIDE`"

The target protein not only selects the `FASTA` file sequence to use, but also selects the line from the `topDownDataFile` that will be read and used as data for this scoring module during a run.

C.4.5 The fragmentScoring Module Data File

This file provides data from experiments that measure and/or identity peptides, such as bottom-up experiments. Each row in this file represents a peptide that has been associated

with a protein through some external matching process.

Columns

Protein: The name of the protein associated with this row. Only rows with a protein name matching the target `proteinName` specified in the config file will be used.

Start: The AA in the canonical sequence that the first AA of the peptide aligns to, starting with the first AA of the protein as 0.

End: The AA in the canonical sequence that the last AA of the peptide aligns to, starting with the first AA of the protein as 0.

Observed: *Unused.*

FragMass: The experimental neutral mass for the observed fragment, used by the `deltaProductMass` algorithm. Ignored if using `errorCounting`.

TheoryMass: *Unused.*

AminoSequence: The sequence identified for a peptide. Used by the `errorCounting` algorithm, ignored if using `deltaProductMass`.

ModList: Type and number of any non-null modification on the peptide. If more than one of the same modification occurs in the same peptide the modification will be preceded by a number and a space (like 2 phosphorylation). If more than one type of modification occurs on the same peptide the different entries are separated by a comma and a space. Many peptides will not have entries in this column. Used by the `errorCounting` algorithm and by the novelty scoring component of the `deltaProductMass` algorithm.

Score: *Unused.* A score specified by MS/MS software giving the quality of a peptides data.

Example

Protein	Start	End	Observed	FragMass	TheoryMass	PpmErr	Miss	
L7-L12	2	5	0.0	471.5539	0.0	6	0	...
L7-L12	6	30	0.0	2704.1820	0.0	5	0	...
L7-L12	31	52	0.0	1997.2345	0.0	4	0	...
L7-L12	53	60	0.0	946.1126	0.0	3	0	...
L7-L12	61	66	0.0	512.5664	0.0	2	0	...
L7-L12	67	71	0.0	510.6780	0.0	1	0	...
L7-L12	72	74	0.0	326.3994	0.0	0	0	...
L7-L12	75	82	0.0	711.8604	0.0	-6	0	...
L7-L12	83	85	0.0	328.3678	0.0	-5	0	...
L7-L12	86	96	0.0	1095.2619	0.0	-4	0	...
L7-L12	97	101	0.0	500.5518	0.0	-3	0	...
L7-L12	102	108	0.0	742.7846	0.0	-2	0	...
L7-L12	109	109	0.0	128.1740	0.0	-1	0	...
L7-L12	110	121	0.0	1226.3476	0.0	0	0	...

(rows continued . . .)

	AminoSequence	ModList	Score
...	SITK	Acetylation	1
...	DQIIEAVAAMSVMDVVELISAMEEK		1
...	FGVSAAAAVAVAAGPVEAAEEK		1
...	TEFDVILK		1
...	AAGANK		1
...	VAVIK		1
...	AVR		1
...	GATGLGLK	Methylation	1
...	EAK		1
...	DLVESAPAALK		1
...	EGVSK		1
...	DDAEALK		1
...	K		1
...	ALEEAGAEVEVK		1

C.4.6 The localizedFragmentScoring module

Tandem MS experiments provide detailed information that potentially provides localization of modifications to specific AA. By matching the modifications and sequence of a guess to those of an peptide experimentally sequenced and aligned to the base protein, the quality of a guess can be evaluated. Much of the information is in terms of what amino acids are not modified. The more amino acids and modifications that match between the aligned MS/MS peptide and a guess, the better that guess should score. The best guess should exactly match all experimental data, although in some cases there will be contradictory data, described in more detail below. The scoring function used to model this is a product over all peptides:

$$S = \prod_i (2^{a_i} \cdot 2^{m_i} + 1)$$

where

$$\begin{aligned} i &= \text{iterator over all matched peptides,} \\ a_i &= \text{count of matching AA for peptide i,} \\ m_i &= \text{count of matching modifications for peptide i,} \\ +1 &= \text{small shift to avoid singularities.} \end{aligned} \tag{C.6}$$

By using a factor of two for each match, the ratio of any two guesses different by 1 error, either an unmatched AA or an unmatched modification, is 0.5. This helps maintain a semi-quantitative interpretation nearly-best guesses relative to the best guess.

It is possible for MS/MS data to contain peptides from multiple isoforms of the target protein. The same issue arrises in bottom-up peptide data.

Parameters

`isLocalizedFragmentScoring` = `false`

Turns scoring by this module on (true) or off (false). By default the scoring of localized sequence fragments is turned off as it requirs experiment-specific data.

`localizedFragmentDataFile` = "`localizedFragment.txt`"

If using `localizedFragmentScoring`, must specify the data file. By default the file "`localizedFragment.txt`" will be searched for, but any name can be specified.

`proteinName` = "`OVER_RIDE`"

The target protein not only selects the `FASTA` file sequence to use, but also selects the line from the `localizedFragmentDataFile` that will be read and used as data for this scoring module during a run.

C.4.7 The localizedFragmentScoring Module Data File

This file provides data from experiments that localize a given modification to a specific amino acid in the protein, or that identify a given amino acid as unmodified, such as MS/MS experiments. Each row represents sequence and/or modification position information from one peptide

Columns

fragmentNum: A continuous sequence of integers, starting with 1, for each fragment of MS/MS data associated with a given protein. If more than one protein is described in the same data set, this should start over from 1 with each.

Protein: The name of the protein associated with this peptide sequence. Only rows with a protein name matching the target `proteinName` specified in the config file will be used.

AminoSequence: The experimentally determined sequence for a peptide.

ModList: The location and type of any non-null modification on the peptide, starting with the first AA of the peptide being 0. Must be within a pair of parenthesis. If more than one modification occurs on the same peptide they are listed sequentially, separated by a comma and then a space, both within the parenthesis set. Most peptides will not have entries in this column.

InitAligntPos: Which position in the canonical sequence the first AA of the peptide aligns to, starting with the first AA in a protein numbered 0.

Score: *Unused* A score specified by MS/MS software giving the quality of a peptides data.

Example

fragmentNum	Protein	AminoSequence	ModList	InitAlignPos	Score
1	L7-L12	SITK	(0|Acetylation)	1	1.0
2	L7-L12	SITK		1	1.0
3	L7-L12	DQIIEAVAAMSVMDVVELISAMEEK		5	1.0
4	L7-L12	FGVSAAAAVAVAAGPVEAAEEK		30	1.0
5	L7-L12	TEFDVILK		52	1.0
6	L7-L12	AAGANK		60	1.0
7	L7-L12	VAVIK		66	1.0
8	L7-L12	AVR		71	1.0
9	L7-L12	GATGLGLK	(7|Methylation)	74	1.0
10	L7-L12	EAK		82	1.0
11	L7-L12	DLVESAPAALK		85	1.0
12	L7-L12	EGVSK		96	1.0
13	L7-L12	DDAEALK		101	1.0
14	L7-L12	K		108	1.0
15	L7-L12	ALEEAGAEVEVK		109	1.0

C.4.8 The modTypeScoring module

Some modifications are more common than others. This scoring module attempts to evaluate how commonly found a given set of modifications is. Guesses containing modifications that are rarer than average will score lower, guesses that contain modifications that are more common than average will score higher. A perfect scoring guess will contain only the most common modification pattern The scoring function used to model this is:

$$S = \prod_i (f(m_i) \cdot (1 - \alpha) + \alpha)$$

where:

$$\begin{aligned} i &= \text{iterator over all non-null adduct modification positions,} \\ m_i &= \text{modification type (non-null) at position } i, \\ f(m_i) &= \log_{10} \text{normalized frequency of modification} m, \\ \alpha &= \text{scaling factor} = 0.9. \end{aligned} \tag{C.7}$$

Exactly how different modifications are scaled relative to one another is important only for comparing near-best guesses. There is no obvious guideline for what a mistake should means in this context, especially since the relative probability of modification depends on context (Archaea, prokaryote, or eukaryote? Membrane, cytoplasm or nuclear?) Additionally, most sources for such information will suffer from ascertainment bias, containing skewed distributions based on what was interesting or easy to study. With this in mind, $f(m_i)$ flattens the true probability distribution by taking the $\log_{10}$ of the relative frequency, and a scaling factor is applied such that the lowest value is α and the highest value is 1.

Parameters

`isModTypeScoring` = `true`

Turns scoring by this module on (true) or off (false). By default scoring by modification type is turned on, using the default properties as specified below.

`modTypeDataFile` = "`modType.txt`"

If using `modTypeScoring`, must specify a data file to read. By default the file "`modType.txt`" will be searched for, but any name can be specified. This file must contain an entry for each modification searched for, as listed in the "`modData.txt`" file.

C.4.9 The modTypeScoring module data file

Columns

PTM_Type: The full name of the modification the row provides data on. All modifications searched for, as defined by the `modData` file, must be listed.

Count: Counts the number of times this modification was found in the source data set. A minimum of 1 count must be used.

Example

```
PTM_Type          Count
Acetylation        2071
Amidation          2150
Deamidation          38
Farnesylation        62
Formylation          32
Oxidation          1074
Methylation         746
Myristoylation      113
Palmitoylation      222
Phosphorylation   22500
Selenocysteine        2
```

C.4.10 The modLocationScoring module

Each modification has a different pattern of amino acid preference. Phosphorylations are usually found on serine and threonine, for instance. This module represents the log-scaled likelihood of a given modification being found on a given amino acid. The scoring function used to model this is:

$$S = \prod_i (f(a_i, m_i) \cdot (1 - \alpha) + \alpha)$$

where:

$$\begin{aligned} i &= \text{iterator over all non-null adduct modification positions,} \\ a_i &= \text{amino acid at position } i, \\ m_i &= \text{modification type (non-null) at position } i, \\ f(a_i, m_i) &= \log_{10} \text{normalized frequency of modification } m_i \text{ on amino acid } a_i, \\ \alpha &= \text{scaling factor} = 0.5. \end{aligned} \tag{C.8}$$

The intent is for one error to equate to a significantly improbable amino acid location set and a factor of two difference in score. This is very difficult to implement as the total number of modifications affects the results, and what constitutes a significantly improbable location set does not have any obvious context-independent definition.

Parameters

`isModLocationScoring` = `true`

Turns scoring by this module on (`true`) or off (`false`). By default scoring of a modification's amino acid preference is turned on, using the default properties as specified below.

`modLocationDataFile` = "`modLocation.txt`" If using modLocationScoring, must specify a data file to read. By default the file "`modLocation.txt`" will be searched for, but any name can be specified. This file must contain an entry for each modification searched for, as listed in the "`modData.txt`" file.

C.4.11 The modLocationScoring module data file

This file provides data from database surveys or other sources that define the average AA preference for each adduct modification Each row represents a modification and the number of times (non-normalized) it was counted in a data set on each of the 20 standard AAs. Data used in the default data file was taken from (Lee et al., 2006).

Columns

PTM_Type: The full name of the modification the row provides data on. All modifications searched for, as defined by the modData file must be listed.

A, R, . . . , V: The counts for the number of times a modification was found in a data set on the amino acid. A "-" is interpreted as 0. All positions will have a number of pseudocounts (by default 1) added to them to allow for rare modifications that might not have made the data set. If a modification is truly not possible, then it should be flagged with -1 which will essentially prevent PIE form allowing the specified modification/amino-acid combination.

Example

PTM_Type	A	R	N	D	C	G	E	
Acetylation	424	7	-	6	5	60	10	...
Amidation	431	52	106	3	73	127	11	...
Deamidation	-	-	30	-	-	-	-	...
Farnesylation	-	-	-	-	62	-	-	...
Formylation	-	-	-	-	-	1	-	...
Oxidation	-	2	11	10	-	-	-	...
Methylation	10	180	22	-	40	-	29	...
Myristoylation	-	-	-	-	-	108	-	...
Palmitoylation	-	-	-	-	210	1	-	...
Phosphorylation	-	0	-	19	3	-	-	...
Selenocysteine	-1	-1	-1	-1	2	-1	-1	...

(rows continued ...)

	(PTM)	Q	H	I	L	K	M	F	
...	(Acet)	-	-	-	-	792	240	-	...
...	(Amid)	21	14	74	358	51	83	398	...
...	(Deam)	8	-	-	-	-	-	-	...
...	(Farn)	-	-	-	-	-	-	-	...
...	(Form)	-	-	-	-	3	28	-	...
...	(Oxid)	-	-	-	-	106	-	-	...
...	(Meth)	22	14	0	7	407	4	6	...
...	(Myri)	-	-	-	-	5	-	-	...
...	(Palm)	-	-	-	-	9	-	-	...
...	(Phos)	-	41	-	-	-	-	-	...
...	(Sele)	-1	-1	-1	-1	-1	-1	-1	...

(rows continued ...)

	(PTM)	P	S	T	W	Y	V
...	(Acet)	14	432	64	-	2	15
...	(Amid)	49	37	38	33	72	119
...	(Deam)	-	-	-	-	-	-
...	(Farn)	-	-	-	-	-	-
...	(Form)	-	-	-	-	-	-
...	(Oxid)	779	17	18	5	124	2
...	(Meth)	5	-	-	-	0	-
...	(Myri)	-	-	-	-	-	-
...	(Palm)	-	0	2	-	-	-
...	(Phos)	-	16590	3472	-	2375	-
...	(Sele)	-1	-1	-1	-1	-1	-1

C.4.12 The modCountScoring module

Some proteins are not modified, and many that are have only a few modifications. This module implements the idea that the best guess will have some specified average number of modifications, and all guesses with more or less modifications then that score worse. The scoring function used to model this is similar to the intact experimental model:

$$S = \frac{1}{|A_e - A_t| + A_\delta}$$

where:

$$\begin{aligned} A_e &= \text{Expected count of modifications,} \\ A_t &= \text{Theoretical count of modifications (from the guess),} \\ A_\delta &= \text{Expect } \pm \text{ count of modifications 50\% of the time.} \end{aligned} \tag{C.9}$$

The intent is for a factor of two change to correspond to one error, equal to a deviation from the expected number of modifications by more than a specified amount. With the defaults set here, this is every 10 modifications.

Parameters

`ismodCountScoring` = `true`

Turns scoring by this module on (true) or off (false). By default scoring by total number of modifications is turned on, using the default properties as specified below.

`modRate` = `0.0`

How many adduct modifications are expected, on average. The default is zero, which works well for many situations.

`modDelta` = `10.0`

The one-sided 25% range in then number of modifications expected. Sets the distribution so that 50% of the time the number of modifications will be `modrate` ±

`modDelta`, bounded on the low end by 0. Using a value of 10 as a default allows for a relatively large number of modifications, but not an unreasonable number.

C.4.13 The cleavageScoring module

In general most proteins have only a few cleaved amino acids, with zero cleavages being the most common. Signal peptides are not considered, and are the subject of another planned scoring module. Cleavages from each end are scored independently, then these scores are multiplied together to obtain the total cleavage score for a guess. The scoring function used to model cleavage from each end is the same:

$$S = a \cdot b_{n-1}$$

where:

$$\begin{aligned} a &= \text{Factor for first AA loss from this end,} \\ b &= \text{Factor for each AA loss after first from this end),} \\ n &= \text{Total number of AA cleaved from this end.} \end{aligned} \tag{C.10}$$

The intent is that an error, corresponding to a factor of two difference in score, this should equate to some number of cleaved amino acids. Defaults give one error for every three or so amino acids cleaved.

Parameters

`isCleavageScoring = true`

Turns scoring by this module on (`true`) or off (`false`). By default cleavage scoring is turned on, using the default properties as specified below.

`cLoss = 0.8`

The factor a from above, for the c-terminal end.

`cLossMore = 0.8`

The factor b, from above, for the c-terminal end.

`nLoss = 0.9`

The factor a, from above, for the n-terminal end. The default is higher than for c-terminal cleavages as n-terminal methionine cleavages are relatively common.

`nLossMore = 0.8`

The factor b, from above, for the n-terminal end.

C.4.14 The ruleScoring module

A rule identifies a pattern than might occur in the guess, and a factor that evaluates how good the guess is in light of the number of times that pattern is detected. When a more complete model does not yet exist or is difficult to create, low hanging fruit rules can be use. For example rather than a conditional model for the probability of one modification given others, rules for specific pairs or repeated occurrences of modifications can be used. As implimented, this scoring function looks like:

$$S = \prod_i R_i = \prod_i a_i n_i$$

where:

$$\begin{aligned} i &= \text{iterator over each rule or pattern to score,} \\ R_i &= \text{the } i\text{th rule,} \\ a_i &= \text{Factor applied for each application of a rule } i. \\ n_i &= \text{Number of times the rule } i \text{ applies to a guess.} \end{aligned} \tag{C.11}$$

The intent is that each rule causes a factor of two change in the score between two guesses when one error has occurred. Factors should be set accordingly.

Parameters

`isRuleScoring` = `true`

Turns scoring by this module on (true) or off (false). By default all rules are enabled with default parameters.

`RuleParameters` = "`amidDeam = 0.5, nTermAcet = 2.0, polyAmid = 0.5, polyDeam = 0.5`"

This parameter specifies the factor to be used for each rule, as a quoted block of comma separated `key` = `value` pairs (may wrap lines). To turn off a rule, set it to a factor of 1.0. Rather than specify each parameter separately, they are all specified in a single list. The rules are:

`amidDeam` = `0.5`

For every pair of amidation and deamidation modifications in the guess, modify its score by this value. By default treat each as an error.

`polyAmid` = `0.5`

For every amidation beyond the first, treat this as an error.

`polyAmid` = `0.5`

For every deamidation beyond the first, treat this as an error.

`nTermAcet` = `2.0`

If there is an acetylation in the guess, and it is on the n-terminal amino acid, treat this as correct (or rather an error if not put there).

C.5 Output Files Reference

C.5.1 Overview

Three output files are generated by PIE:

- `summaryFile` - The answers, the best scoring guess from each search in a run set.
- `detailFile` - Answers sampled regularly during each search in a run set.
- `logFile` - A file where PIE reports all kinds of status messages, usually of little user interest, with the exception of the random seed that allows repeating runs exactly and the time being taken for each run in a run-set.

Parameters

The following properties affect all output files:

Parameters

`outputDir` ="`./`"

The directory to produce output into, by default the current directory.

`isAutoOutputDir` = `true`

Generate a new directory in the output directory for each run of PIE, named for the date and time when PIE ran.

C.5.2 The summaryFile

This file provides the answers, the best guesses obtained during a run by PIE. One line in this file corresponds to one search for a best (highest scoring) guess. The number of searches to make per run is specified by the `runcount` property.

Parameters

`summaryResultsFile` = "`pieSummary.txt`"

Name for the file where PIE outputs the final, best scoring guess for each sampling walk of a run, appending guesses as they are obtained.

Columns

Steps: The total number of steps taken during the search, as set by the maxSteps property

BestStep: The step number on which this guess was first seen.

BestScore: The score for this guess.

Best . . . Score: The component score for this guess. Each scoring module reports the score it obtained for this guess in a column named for it. For example, the column Best*IntactMass*Score contains the score returned by the `IntactMass` module and evaluates the intact mass data against this guess.

CleavedLength: The total length of this guess, shorter than the length of the canonical protein if there are any cleaved AAs.

RawLength: The length of the canonical protein, without any cleavages.

NTerm: The position of the N terminus of this guess, relative to the canonical protein's first AA, starting at 1.

CTerm: The position of the C terminus of this guess, relative to the canonical protein's first AA, starting from 1 .

TotalAverageMass: The average mass for this guess.

TotalMonoMass: The monoisotopic mass for this guess.

ModCount: The total number of adduct modifications. For each modification another three columns will be reported, giving the ModPos, AA, and ModName in that order.

ModPos: The position for a modification, relative to the canonical proteins first AA, starting from 1. See also columns AA and ModName.

AA: The AA modified. See also columns ModPos and ModName.

ModName: The name of the modification. See also columns ModPos and AA.

Example

```
Steps   BestStep          BestScore   BestIntactMassScore  BestLocalizedFragmentScore
10000       1070  5.643437533751E70        47.12517580111          4.4171176619459E71
10000       4766  4.749163449850E67        40.94990681763          2.7606985387162E70

  BestFragmentScore BestModLocationScore  BestCleavageScore  BestModCountScore
0.0038333049159276     1.0                              0.9    0.8333333333333
3.2679504054652E-4     0.44302694950605                 0.9    0.7142857142857

BestModTypeScore  BestRuleScore  CleavedLength  RawLength  NTerm  CTerm
 0.9430118309690            1.0            120        121      1    120
 0.9027242997591            0.5            120        121      1    120

   TotalAvgMass     TotalMonoMass  ModCount  ModPos   AA     ModName
12220.088999999   12212.498369999         2       4   K  Acetylation
12220.092199999   12212.509602999         4       4   K  Acetylation

ModPos  AA        ModName  ModPos  AA    ModName   ModPos  AA   ModName
    81   K    Methylation
    22   L      Amidation      77   G  Oxidation       79   G Amidation
```

C.5.3 The detailFile

This tab delimited file outputs the current working guess at specific intervals during the search rather than the single best result found at the end. How often this is done is determined by the `everyN` property. It has the same column format as the detail file, except it does not report the individual module-based scores for a guess, and it does not have column headers. This file can be used to observe how the McMC search converges from an initial bad guess to a final good one. If PIE fails to converge for a given data set, this file provides details to help diagnose the problem.

Note that PIE holds all these in memory until the summary guess is reached, so if too many interim samples are taken, PIE may run out of memory.

Parameters

`detailedResultsFile` = "`pieDetails.txt`"

Name for the file where PIE outputs intermediate guesses proposed during each sampling walk of a run, appending guesses at the end of a walk.

Columns

Column names are not provided in this file. Headers show in the example file are provided to make description of the columns easier. Ellipses (. . .) indicate where lines from the example file are not shown.

1. Steps: The total number of steps taken during the search, as set by the maxSteps property.
2. BestScore: The score for this guess.
3. Len: The total length of this guess, shorter than the length of the canonical protein if there are any cleaved AAs.

4. Raw: The length of the canonical protein, without any cleavages.

5. N: The position of the N terminus of this guess, relative to the canonical protein's first AA, starting at 1.

6. C: The position of the C terminus of this guess, relative to the canonical protein's first AA, starting from 1.

7. TotalAvgMass: The average mass for this guess.

8. TotalMonoMass: The monoisotopic mass for this guess.

9. ModCount: The total number of adduct modifications. For each adduct modification another three columns will be reported (ModPos, AA, and ModName, in that order).

10+ ModPos, AA, and ModName: The modification position (on the uncleaved protein), the AA modified, and the adduct modification name, respectively.

Example

Steps	BestScore	Len	Raw	N	C	TotalAvgMass	TotalMonoMass	ModCount
0	6.198605E64	121	121	0	120	12295.2242999	12287.5126449	0
2500	3.608771E65	118	121	3	120	12216.2233999	12208.6013809	2
5000	3.608771E65	118	121	3	120	12216.2233999	12208.6013809	2
...								
35000	2.681259E65	118	121	3	120	12216.2233999	12208.6013809	2
37500	6.284537E67	120	121	1	120	12220.0889999	12212.4983699	2
40000	1.111083E68	120	121	1	120	12220.0889999	12212.4983699	2
...								
50000	1.111083E68	120	121	1	120	12220.0889999	12212.4983699	2
LAST	1.111083E68	120	121	1	120	12220.0889999	12212.4983699	2
0	6.198605E64	121	121	0	120	12295.2242999	12287.5126449	0
2500	2.234627E66	120	121	1	120	12206.0623999	12198.4827199	1
5000	2.234627E66	120	121	1	120	12206.0623999	12198.4827199	1
...								
47500	4.310145E71	120	121	1	120	12220.0889999	12212.4983699	2
50000	4.310145E71	120	121	1	120	12220.0889999	12212.4983699	2
LAST	4.310145E71	120	121	1	120	12220.0889999	12212.4983699	2

(Steps)	(ModCount)	ModPos	AA	ModName	ModPos	AA	ModName
0	0						
2500	2	4	K	Palmitoylation	80	L	Methylation
5000	2	4	K	Palmitoylation	78	L	Methylation
...							
35000	2	4	K	Palmitoylation	74	G	Methylation
37500	2	81	K	Methylation	89	S	Acetylation
40000	2	15	S	Acetylation	81	K	Methylation
...							
50000	2	15	S	Acetylation	81	K	Methylation
LAST	2	15	S	Acetylation	81	K	Methylation
0	0						
2500	1	2	I	Acetylation			
5000	1	2	I	Acetylation			
...							
47500	2	1	S	Acetylation	81	K	Methylation
50000	2	1	S	Acetylation	81	K	Methylation
LAST	2	1	S	Acetylation	81	K	Methylation

C.5.4 The logFile

The `logfile` is where the PIE makes comments about what it is doing as it goes along. Most of this information is of little use unless trying to debug problems. The main user interest in the log-file is the random seed. This can be used to exactly regenerate results. This file can also be used to verify that the correct input data was processed, or to check which property settings were used after the fact. However, the format is unspecified and may change completely. Currently an attempt is made to provide a label for each message giving its priority, but this is not guaranteed.

Parameters

`logFile` = "`pie.log`"

Name for the log file where PIE will output errors, warnings, information, and debug information, as selected by the `logFilterLevel`.

`logFilterLevel` = "`DEBUG_LOW`"

What information to output to the log. Allowed values are "ERROR", "WARNING", "INFO", "DEBUG", and "DEBUG_LOW" Also contains the random seed used.

`startSeed` = `0`

The random seed to use, if 0, will generate one automatically.

`# startSeed` = "`50, -10, 64, 16, 23, 64, 80, -47`"

An example of the format for the random seed, a comma-separated list of 8 signed 1-byte numbers. Spaces are not significant.

Example

The line reporting the random seed will look like:

```
[WARN] Using self-generated random seed: [67, 81, -116, 43, 94, -109, 15, -46]
```

Appendix D

Tutorial Walk Through

This tutorial uses version 0.3.5 of PIE.

D.1 Overview

To illustrate the use of the PIE, you and I will perform a step by step analysis of a synthetic example of an isoform of the L16 ribosomal protein. This tutorial focuses on what to do to use PIE to analyze data, leaving out much of the why. A more succinct description is available as part of a book, Bioinformatics for Comparative Proteomics (Jefferys and Giddings, 2011A). For more details on the rational behind different steps, see Chapter 3, Methods. I assume that you already have PIE installed, and that necessary data files provided with the distribution (in the demo directory demo/L16) are available. PIE and installation information can be download from `http://pie.giddingslab.org/`.

Running Pie take place in four phases, in summary the phases are:

Setup Data to be integrated is formatted and the configuration file PIE uses in lieu of command line parameters is set up.

Convergence Several runs of differ lengths are used to find the highest scoring answer, as well as estimate the relation between run length and the frequency with which the best scoring answer is found by PIE.

Profiling Using the convergence information, a large run-set is generated to profile the distribution of best and nearly best answers and their scores.

Interpretation Analysis of the profile provides relative information about how good the best and near-best answers are and how good the different data sets are. This is

especially important when there are multiple isoforms or is some of the data is bad or contradictory.

The remainder of this tutorial is separated into four sections, one for each phase. The sections begin with a general description of the phase and its goals, and then a sequence of numbered steps provide instructions to replicate an analysis of the modifications on a theoretical L16 protein. I describe each default data file and configuration parameter in terms of how it is used as I go. More comprehensive information on the these files and parameters can be found in the users manual (Appendix C).

D.2 Setup

Setting up a the PIE run involves collecting and editing data files needed as input to PIE, and then setting `key=value` parameters in the configuration file used to control how pie runs. For the most part, this example will use pre-prepared data files provided with the PIE distribution that already contain the necessary input information. Similarly a template configuration file is available with most of the configuration parameters set correctly.

1. Create an experiment directory to work in.

It is important to keep organized when running the PIE, since multiple input and output files are involved. The simplest way to use PIE is to dump all needed files into the same directory, and that is the example presented here. A hierarchical structure allowing for data to be shared across multiple proteins is supported by PIE, and is more convenient in more extensive analysis covering multiple proteins (e.g. see section 4.3, Analysis of L7/L12 Ribosomal Extracts). For this example I assume a project investigating ribosomal proteins from E. Coli K12, specifically the L16 protein, and even more specifically an isomer with an intact mass of about 15,222 Da. I use only the last directory for data, but create the full hierarchy anyway. The full path for the target directory should be the one used on your system; on my system the directory I used is:

/Users/jefferys/ribosomeProject/L16/A-15222

2. Copy and edit the data files.

The molecule and template data files from the distribution `pie/data/` directories need to be copied to the experiment sub-directory and modified as needed.

2.1. Copy the "pie/data/molecule/..." files to "A-15222/".

These files provide mass information to the PIE about component molecules (water, amino acids, and modifications).

molecules.txt: No changes are needed.

aminoacids.txt: No changes are needed.

modifications.txt: This could be edited to change the searched for modifications, but the default list is fine for this example. The list contains every modification that PIE will consider.

2.2. Edit the copied "molecule/..." files.

No editing is needed.

2.3. Copy the "pie/data/prior/..." files to the directory "A-15222/".

These files provide data to pie to use when applying prior belief and average case expectations.

modType.txt: This file is required by the `modType` data module. I will use the default template file, except I will reduce the weight of phosphorylation by 10x, from 22,500 to 2,250, to represent my belief that ribosomal proteins are significantly less likely to have a phosphorylation than is typical for most proteins. This is an example of using domain-specific information to modify average expectation to match a more specialized context.

modLocation.txt: This file is required for the `modLocation` data module. If I knew more about the kinds of enzymes that modify ribosomal proteins, I might use such domain knowledge to adjust the background distribution to better fit the narrower ribosome-specific context. However, for this example I will use the defaults.

2.4. Edit the copied "prior/..." files.

Edit the copy of the `modType.txt` and reduce the weight for `phosphorylation` from `22500` to `2250`. Be careful not to change anything else, especially the tab delimiter that exists between "phosphorylation" and "22500". Don't use comma's either!

2.5. Copy "pie/data/experimental..." files to "A-15222/".

Since I have intact, peptide mass, and peptide sequence data, I will be using all three experimental data scoring modules (`IntactMassScoring`, `FragmentScoring`, and `LocalizedFragmentScoring`) and their data input files. The `FASTA` file containing the canonical sequence for our protein is also provided in this step.

targets.fasta: PIE requires a FASTA file containing the sequence of the target protein. My target protein, `L16A-15222`, is the name of an entry in the provided `targets.fasta` file. I include this as an experimental data file and not a molecule file, as it could change with every experiment. In future versions of PIE, better protein naming options will be supported, but for now all names in all files must match the protein name as given at the beginning of a definition line in the fasta file.

intact.txt: This file gives the value I am using for an experimental intact mass, 15222.1977, and information about the error in the measurement, here within ± `10 ppm`.

fragments.txt: The peptide mass data would usually comes from a subset of trypsin digest fragments attributed to a protein by precursor mass matching. For this experiment I simulate moderate coverage of the target L16 isomer, including peptides showing an oxidation adduct modification. (See Fig. 7, "*L16 Theoretical Data*") However, there is no coverage of the N-terminal end, meaning there is no peptide information indicating the met trun-

cation or the acetylation, and I have also failed to detect a methylated peptide. Indeed, only 1 of 4 modifications is present in this data.

localizedFragments.txt: Close analysis of putative MS/MS fragmentation data has been used to provide protein sequence ladders. Such an examination has pinned down the exact location of the oxidation modification. Non-modified fragments specify regions where no modifications are expected. Note that there is still no fragment data for the existence of the methylation or acetylation adduct, nor for the n-terminal met truncation.

2.6. Edit the copied "experimental/..." files.

No editing is needed.

3. Copy and edit PIE's configuration file.

PIE uses a configuration file to tell PIE everything it needs to know to run including where to find that data, how to look for answers, and how to report the results. As a result, only this file is needed as a parameter when running PIE, but making sure this file has all the correct information is very important.

3.1. Copy "pie/data/default.properties" to "A-15222/".

default.properties: This template properties file will need to be edited every time the PIE is run. These changes are described in the next step

The parameters used in the file are discussed in more detail in the user manual (Appendix C). I will go through the file by sections, detailing the parameters and changes relevant to this analysis of L16 data. The three sections are:

Data and data models: Defines what data types to use, how to score them, and where to read data from.

McMC configuration: Used to set tuning parameters for defining how PIE does its sampling. Normally default are fine.

Run and reporting: Determines how deep and how many separate searches PIE makes, along with how to report results.

3.2. Edit the "Data and data models" section of the properties file.

3.2.1. Set up the "Data directories" subsection.

I have copied all the data into one directory, so I set the `defaultDataDir` parameter to the full pathname for the directory, and leave the others at their default (unspecified) values.

`defaultDataDir` = "`/Users/jefferys/ribosomeProject/L16/A-15222`"

Note: Make sure to replace "`/Users/jefferys`" with your own path.

3.2.2. Set up the "Molecular data" subsection.

For this example all three molecular data filenames are correct and can be left unchanged, as "`molecules.txt`", "`aminoacids.txt`", and "`modifications.txt`", the default names match the files set up above.

3.2.3. Set up the "Experimental data" subsection.

In general you can provide a copy of your target protein in a `FASTA` file named something other than "`targets.fasta`" by specifying the new file via the `proteinFastaFile` parameter, but I will use the default.

The target protein name must be set to match one of the protein names given in this `FASTA` file of targets. This example uses the protein sequence for L16-A, so the `targetProteinName` parameter must be set to this value.

I am using all three experimental scoring modules (intact, fragment, and localized-fragment scoring). Each will need to have their respective boolean flag set to true and need to specify the name of the input file to read. The fragment scoring module requires an additional parameter specifying the algorithm to use in scoring the data. See also 5.2.2 Evaluating a peptide data set

All of these values are correct by default for this experiment, so in summary the changes are:

`targetProteinName` = "`L16-A`"

3.2.4. Set up the "Prior data" subsection.

I am using all five prior data scoring modules (mod count, mod type, mod location, cleavage, and rule-based scoring). All five will need to have their respective boolean flag set to true and two requiring input files (mod type and mod location). This is all correct by default.

The remaining three scoring modules have no data file to read, but instead have parameters that are configured as follows:

For the `modCount` model I only know of one possible modification, an oxidation. However the intact mass does not reflect such a simple answer, so I adjust the expected number of modifications to `2.0`. There is no need to be exact when setting this parameter as I have good intact data (see also 5.2.4 Setting the number of modifications). Setting it to 1.0, or 0.0 would still work, but as with any modeling process the more accurate and consistent the data provided to the PIE, the better.

I have no reason to suspect a large number of modifications, so I set the range (via `modDelta`) to `7.0`, which allows `2.0` $\pm$ `7.0` modifications with reasonable probability, when supported by other data. The choice for `7.0` as a range is based on the moderate coverage of the peptide without other modifications, the small number of modifications already seen, and a bias towards believing that if the protein has a large number of modifications, I would know.

I have no information about cleavages to add to the cleavage scoring model, so I use the default values. These are intended to open cleavage (cleave the first AA) at the n-terminal end with moderate efficiency, to open the c-term end more rarely, and then to follow through with additional cleavages moderately often from both ends if opened.

For the `ruleScoring` module, I also leave the rules as they are by default. I don't expect to have both amidation and deamidation modifications at the same time, and I expect that if acetylation occurs, it is often found on the N-terminal amino acid.

In summary, the changes are:

`modRate = 2.0`

`modDelta = 7.0`

3.3. Edit the properties file "McMC configuration" section.

Seldom will any changes be necessary in this section. There is nothing to change in this experiment, so I use only default values.

3.4. Edit the properties file "Run and reporting" section.

This sections is the most commonly edited portion of the parameters file. Since obtaining a full answer profile from the PIE requires ensuring the length of a run is sufficient to obtain convergence, several small profiles at different run lengths are generated to determine this. After the optimal length is determined, the complete profile can then be produced. This requires multiple runs of the PIE with different numbers of searches and/or different run length. Each requires adjusting a couple of parameters in this section. Besides the basic run parameters, this section also controls reporting details.

3.4.1. Set up the "Reporting parameters" subsection.

I set the top level `outputDir` to be the experiment directory. As `autoOutputDir` is `true`—its default value—each PIE run will generate its own subdirectory in the experiment directory.

I can leave the summary and detail output file names at their default values as there is no chance of them over-writing previous results and I will also leave the log file at its default setting, so the only change is:

`outputDir` = "`setspace.sty/Users/jefferys/ribosomeProject/L16/A-15222`"

Note: Again, make sure to change "/`Users/Jefferys/ribosomeProject`" to something that matches your configuration.

3.4.2. Set up the "Runtime parameters" subsection.

Given that the PIE takes several minutes or more to generate useful profiles, I start instead with a short test run, designed to flush out any errors made during the process of setting up a run. I set up a test run that will only take a few seconds and create a profile with `2` entries (set by the `runCount` parameter), each of `10,000` steps (`maxSteps` parameter). During this process, the PIE will output current status entries to the console every `2,500` steps (`consoleUpdate`), and update the detail results file every `1,000` steps (`everyN`), resulting in 4 console updates and 10 detail file entries for each of the two runs. Summarized, the parameter settings required to do this are given below. *Note*: No commas are allowed in values.

`everyN` = `1000`

`consoleUpdate` = `2500`

`maxSteps` = `10000`

`runCount` = `2`

4. Rename the properties file.

Since the PIE will run multiple times using different settings and each time need different properties file settings, it is useful to rename the control file to something describing the run, such as "`test-2atE4.properties`".

5. Perform a test sampling run with PIE.

From the experiment directory with the properties file and data run PIE. If you have set up pie as an executable this will be:

```
pie ./test-2atE4.properties
```

otherwise this will be:

```
pie -jar /path/to/pie.jar ./test-2atE4.properties
```

If everything works correctly, the PIE will run for a few seconds, writing output to the screen as it goes. Three output files will be created in a time-stamp directory inside the experiment directory, named something like `2009_12_17__18_23_35_951`.

The "`pieSummary.txt`" file should contain 2 lines, with a separate `best...` scoring columns for the total score and for each data model used. Here, this is 1 total + 4 prior + 3 experimental = 8 columns. The "`pieDetails.txt`" file should contain 2 sets of 10 lines each. The "`pie.log`" file is useful if errors occur and the expected results are not generated.

6. Rename the time-stamp directory.

Since this is a test run, and does not contain useful results, you could delete it. If you wish to keep it, it is probably a good idea to rename the time-stamp directory to match the properties file, to `test-2atE4` to allow tracking what came from where.

D.3 Convergence

Simulated annealing by MCMC is a stochastic search method that finds global optima based on a guided tour through a landscape of candidate answers. As a stochastic method, how long the search for an answer takes varies. Once the PIE has been successfully configured and the test run is successful, the next phase is to find the convergence length. This involves making a small run containing about 10 replicates or so at one run length, then making another run of 10 at a longer run lengths, and continuing until convergence is apparent.

For example, if the PIE is run for 10 replicates and the highest scoring modification scenario found is found multiple times (say 3 of the 10 replicates), this may be the best answer. If another run with longer searches produces the same high-scoring scenario more often then the previous run (e.g. 5 of 10 searches), this is reasonable evidence that the best answer has been found.

In contrast, it may be that there is no consistency in the highest scoring results upon the first run. This probably means longer runs are needed. With long enough runs a best-scoring scenario should eventually emerge. The more steps it requires to find this best scoring value, the weaker the data, However, there is a chance that if the data is insufficient to resolve any answer and no best scoring answer is possible. This is often indicated by the best scoring answer being the uninformative prior, usually a completely unmodified protein. In contrast, if convergence is obtained quickly at low run lengths it likely indicates that the data very strongly supports the resulting solution.

It is important to keep separate the two different ways I will use PIE: First to determine convergence, and secondly to generate a profile of candidate answers. Hopefully the determination of the convergence values will be automated in the future, but for now this must be performed manually as described below. The following section will then describe the profiling process.

1. Copy the properties file.

The properties file created during setup will be a template for each of several runs. Copy it to a representative name, using the number or replicates and the length in the filename is a good idea as it makes later reference easy. I will use "`conv-10at1e5.properties`" to indicate 10 searches each 100,000 steps long.

2. Modify the run parameters in the new properties file.

I want to do 10 replicates of 100,000 steps, and need to set the appropriate parameters. This also means setting two additional parameters that tells pie how often to check back in during what might be a long run. These same four runtime properties will be adjusted for each different convergence profile, and also for the final candidate solution profile. `RunCount` gives the number of replicates, so I set it to 10, `maxSteps` is the number of iterations, so I set it to 100,000 (no commas!), `everyN` determines how often interim results are output to the details files, I want about ten so I set this to 10,000, and `ConsoleUpdate` describes how frequently interim results are output to the console. Setting this to about 25,000 gives us output often enough to follow. In summary:

```
everyN = 10000

consoleUpdate = 25000

maxSteps = 100000

runCount = 10
```

3. Run PIE.

Run the PIE using the new properties file as the argument, and rename the output directory when done to match the properties file. This is the same as steps 6 and 7 in the setup section previously, except the correct properties file name should be used, and the content of the output files will be different.

Running this convergence test make take up to several minutes on a computer system circa 2011. A new time-stamp directory will created. Some data is output into this directory as the PIE runs, and additional data is added at the end of the run. When complete, the "`pieSummary.txt`" file will contain the best scoring answers from each of the 10 runs.

The file format is a simple tab-delimited text file with a header row, so standard spreadsheet programs can be used to import, view, and manipulate this file. (See also 5.2.8 Supporting R scripts). The `bestScore` column gives the total score of each run. I check to see whether the highest score is repeated. If so, I am on my way to determining the convergence length to use in the full analysis.

4. Execute a longer convergence run.

Repeat the previous two steps to generate a second convergence graph, but set pie to use longer searches, something like 2 to 10 times the previous value. That entails making a new copy of the properties file, renaming it to something like "`conv-10at5e5.properties`", and changing the run parameters to something like:

```
everyN = 50000

consoleUpdate = 25000

maxSteps = 500000

runCount = 10
```

As before, I run pie, rename the output directory to match the properties file, and examine the "`summary.txt`" output file to look for repeated high score values in the `BestScore` column.

5. Keep repeating until convergence is obtained.

Until I obtain two successive profiles where the maximum score is repeated, I will keep increasing the maximum number of steps and generating new profiles. The amount to

increase the step size by matters only in the sense that I am trying to find the convergence length in a small number of guesses without wasting too much time on large guesses. If I make too large a jump, (indicated by going from one run showing little or minimal convergence directly to one showing almost every search having the same maximum value, I can always go back and try an intermediate guess for the number of steps needed.

Fig. 8, "*L16 Convergence*" shows the results from 3 convergence runs for the L16 target. As with any other stochastic process (such as measurement!), the results from the PIE will differ in specifics even for identical input, but on average are consistent. My initial run of 10 at 100,000 steps appears to show convergence, with 4 of the 10 runs having the same high score. Since PIE is stochastic, it won't generally get the same results on repeated runs, so results will vary. I might have had 3, or 5, but probably not 0 or 10.

In the second run of 10 at 500,000 steps, only the top scoring candidate was found, 10 out of 10 times. This is really good evidence that this top scoring candidate really is the best scoring candidate that could possibly be found.

However, For profiling I will need to know the convergence length that has about 2 of the 10 searches finding this global best candidate (see also 5.2.4 The Need for an accurate convergence parameter). I run another set of 10 at 50,000 steps (copying the parameters file, renaming it, and adjusting the same 4 run parameters). This seems to be exactly the correct number of steps, resulting in 2 of the 10 searches having the maximum value.

D.4 Profile

Once the convergence length estimate has been determined, it becomes possible to generate an answer profile. To do this requires simply running a set of 100 or so samples at the 20% convergence rate determined above. About 20 of the profile searches should result in the same high-scoring answer, while the other 80 or so represent a sampling of near-optimal answers. The represented distribution over near-optimal answers provides a great deal of information about the way the data is structured and what kind of answer the best answer is. Sub-optimal answers that score high relative to the best answer indicate answers with less but significant support, including answers that might represent alternate isoforms or answers that can't be distinguished due to missing data or data with insufficient resolving power.

1. Set run parameters to create an answer profile.

Although the goal is different, the process is the same. As before I will create a copy of a property file, name it something useful, such as "`profile-100at5e4.properties`", modify the run parameters, and then run the PIE. For this example, the "`conv-10at5e4.properties`" file is already mostly correct, so I choose to copy that as a template. Then only the `runCount` needs to be changed, from 10 to 100. The other values are already correct:

```
everyN = 5000

consoleUpdate = 2500

maxSteps = 50000

runCount = 100
```

2. Determine if done, or need to try again.

Examine the "`pieResults.txt`" and determine if it is good enough: If the top few candidates of interest are not found several times each, then the run length and/or number of searches will need to be increased.

2.1. If the best scoring candidate is represented too many or too few times, increase or decrease the run length and try again.

The profile I generated using 50k steps does not provide enough replicates for top end, indicating I underestimated the conversion length. Only 2 / 100 answers were the top-scoring result. Why did it miss? As may often be the case when there are modifications without supporting bottom up data, there are several very similar scoring candidates. The scores are close enough that my original corse estimate for the convergence length did not distinguish between them. This larger view indicates a longer search time is required to improve the separation from other very nearly identical answers.

I repeat the previous step and generate another profile with 100 searches, using 100k steps. As before I copy another profile, rename it, edit the run parameters, and run pie, generating an output folder, which I then rename.

The results from this second profile run indicate 100K steps is a good run length to use, and the results are pictured in Fig. 9, "*L16 Answer Profile*" Although I still have only 10% of the answers in the top scoring set, this is 10 actual answers, so it is unlikely that a better result has been missed by chance. The following nearly best answers are also well separated, each presented in its own wide, equal-scoring block. This is sufficient for the example, and I can interpret these results.

2.2. If the top candidate is represented a reasonable number of times, but other similarly scoring candidates have only 1 or 2 replicates, increase the number of searches and try again.

Creating a larger profile, say with 500 runs, increases the resolution. Providing more more searches is like adding pixels to a picture, here widening every scoring band. It may be possible to decrease the search length if running many extra searches. Deciding what size profile to use with what length search in order to obtain appropriate coverage of the near-optimal answers is a choice between deeper coverage of lower scoring answers vs fast coverage of the top answers. Fortunately a wide range of values generates useful results, so it is not critical to pick perfect values, just adequate ones. For profiles involving several modifications, 100 searches works well. If there were 10 different modifications, there would likely be more high-scoring combinations for the same level of coverage and hence require more like 1000 searches to adequately sample from them all. The only cost of picking bad values is wasted computational time. Given the L16 data and scoring modules used, generating the answer profile shown in Fig. 9, "*L16 Answer Profile*" required about 15 minutes; generating a profile with more than 100 searches or more than 100,000 steps per search takes proprtionally longer.

D.5 Interpretation

It has taken quite a bit of work to obtain the answer profile, so it is probably a good idea to step back and check out what was done.

I collected several different kinds of experimental mass spectrometer data derived from a (theoretical) variant of L16 ribosomal protein variant. To this collection of experimental results, I added some general prior knowledge about modified proteins, such as which modifications are more common. I also included information that applied to a specific domain, by specifying a lower likelihood of phosphorylation than average due to the protein being a ribosome component. I then dumped all that information into a directory and used the PIE to put it all together and tell us about the modified protein variant or variants described by this data.

To process and integrate data, I ran PIE in two different ways, first finding a convergence length and then simply running pie with the correct length to find not only the best answer—the modification pattern that is most consistent with the PIE-evaluated data—but also the runner-up choices. Estimating the correct convergence length was not completely successful using quick and short runs of 10 searches each, but after one round of 100 searches, a useful value was produced. The second round of 100 searches using the longer searches generated an answer profile with enough resolution to be interpretable. This answer profile is presented in a compact form in Fig. 9, "*L16 Answer Profile*". Interpreting this information is the subject for the rest of this section.

1. What modifications are present in the highest scoring candidate?

In this example, the highest scoring candidates present a set of 4 modifications—one methylation, one oxidation, one acetylation, and a single n-terminal amino acid loss (methionine). This accurately reflects the exact set of modifications expected from the target, although I would not know that when analyzing actual samples.

2. Is there a consensus set of modifications present in the runner up candidates

In this example, almost all the highest scoring candidates present the same common set of 4 modificationsone methylation, one oxidation, one acetylation and a single n-terminal amino acid loss (methionine).

The presence of these modifications are thus a highly supported predictions for the protein variant. These modifications have been identified despite there being no specific mention of a methylation or acetylation adduct anywhere in the experimental data. Furthermore, the lack of any peptide information about the first 6 AA at the n-terminus allows leaves the possibility of any length truncation from 0 to 6 amino acids. Such truncations would all be equally and fully consistent with the experimental peptide data. However, only the correct 1-AA length truncation is selected. Essentially this is due to the PIE's ability to access the large amount of information contained in an accurate intact mass measurement, solving the combinatorics puzzle of what pieces might be put together to get the given mass. It is also affected by the prior data that pushes guesses towards fewer modifications and shorter truncations.

3. What if there are no consensus or competing high-scoring modification sets?

Comparing the other scoring columns from the "`pieSummary.txt`" file helps determine how well each type of data is supported.

There is one alternate candidate, found only once, that does not have these modifications but still scores relatively highly, with 2 Oxidations, 2 amidations, 1 acetylation, and the same 1 AA n-terminal truncation. It is the highest scoring candidate that is not near identical in score to the top-scoring candidates, being only about 80% as good. This is still a pretty reasonable candidate, so I will need to look at the other scoring columns to see why it scored less. For example, the intact scoring column has nearly identical

scores with the previous example. This previous candidate differs from this one by having a methylation (14.0269 Da) instead of an oxidation (15.9994 Da) and two amidations (-0.9847 Da each) for a net mass of 14.03 Da. This candidate is thus isobaric with the better-scoring candidates, as it is different by only 0.0039 Da, about 0.2 ppm of the intact mass, well within experimental error of 10 ppm.

The worst scoring component is the mod location score, indicating at least one of the modifications in this lower scoring answer is placed on a less common than average choice. This does not really affect a choice for this as a good modification data set, so we are left with this as a reasonable candidate.

One low scoring result with a different set of modifications was commonly found: 1 formylation, 1 acetylation, and 1 n-term cleavage result. It is only about 7% as good as the best one. Its main failing is it doesn't contain an oxidation modification; this conflicts with explicit MS/MS information. Some significant problem *must* be associated with this and every other low scoring candidate, as that is what it means to be a low-scoring candidate. Note that this is only relatively a bad candidate. It still matches top down data very well, and doesn't have 30 phosphorylations, etc. This is still one of best candidates out of the enormous number of *possible* guesses, although its low score indicates it is, relative to other best choices, not so good.

4. Determine if there is one candidate set only, or multiple candidates.

If only one candidate set occurs though all high scoring answers, this is the only candidate set. If there are other candidates besides the best with different modifications that score relatively highly, this means their are either multiple candidates or different isoforms.

In our example, we have two candidates that score well. Without knowing what the target actually was, it is possible the answer to this example involved 2 amidations, 2 oxidations, and an acetylation rather than an oxidation, a methylation, and an acetylation.

Nothing in the data precludes this. Since the set without the amidations scores somewhat higher, if I had to choose just one predictions, this would be it. But that is overly simplistic. The real answer is that the data is suggestive but not conclusive and supports two different alternate isoforms. I need to gather additional evidence, possibly carefully looking for fragment data with amidations or methylations.

The new data could then be added and PIE rerun. If one candidate is supported over the other, he new profile would then score all candidates differently, integrating this new data. One of these two scenarios would then be elevated, the other reduced in score. Part of the design goal for PIE is for it to be an active participant in the process of selecting which experiments or additional data is necessary to nail down the exact modification pattern for an example.

It is also possible that both isoforms are actually present. In this case additional experiments to separate the isoforms could be used to verify that each is actually present.

5. Look at modifications with consensus positions.

In this example, all searches have the PIE placing the n-terminus of the protein after the initial methionine, predicting a 1 AA truncation, and the c-terminus of the protein after the last AA, predicting no n-terminal truncation. These are both correct given the known L16 target. All top scoring answers also correctly specify an oxidation on 107-P for all the answers scoring higher than 50

I only have one example including the amidations. If I believe this to be a viable candidate, I need to run more PIE searches (larger profiles) to add resolution to the bands in the middle of the graph. To keep this tutorial example simple, I will focus only on the best scoring candidate.

6. Evaluate modifications without consensus positions.

Why would PIE fail to find a single best position for a modification? This may be

due to conflicting information from fragment data or missing information.

There is no clear consensus on where to place the methylation or acetylation modifications. Several different positions are suggested for each within all the candidates that scored essentially 100%. If lack of consensus is due to conflicting fragment information, this may be evidence for multiple positional isoforms. This is not the case in the example.

Since the target is known ahead of time, the acetylation is supposed to go on the n-terminus (after truncation of the methionine), and the methylation is supposed to go on 49-R, but unfortunately there is no data telling the PIE this. The intact mass data is no help, as it only depends on what modifications there are, not where they go. The MS/MS information contains data that the PIE can use to localize the oxidation, and also constrains where not to put any other modifications. The fragment data is weaker support, but still guides where not to put and not put modifications in general. However, after taking all this into account there are still many places were the needed adduct modifications could go, and no experimental evidence to chose any position preferentially.

PIE will try to fall back on prior data in this case, because sometimes this would make all the difference. For example, if a selenocystine modification remained to be placed, and there was only one cystine, then the prior data would pick the correct one. For acetylation and methylation, the prior data is not so useful as there are many different amino acids that might be modified that are present throughout the protein. For methylation the prior data first selects lysines (K), and only when it can't place modifications there will it target arginines (R). For the acetylation, PIE tries to guide it to the n-terminus using the rule-based prior that raises the score if it is placed there. Unfortunately the n-terminus of this protein, after truncation of methionine is L. Based on the prior location data, this will be nearly the last place the PIE would put an acetylation, first selecting K, then A then S, etc. One can see this prior at work in Fig. 9, "*L16 Answer Profile*", as different amino acids are picked for methylation and acetylation, and since they both have the same best choice for AA, it swaps the modification on alternate bands.

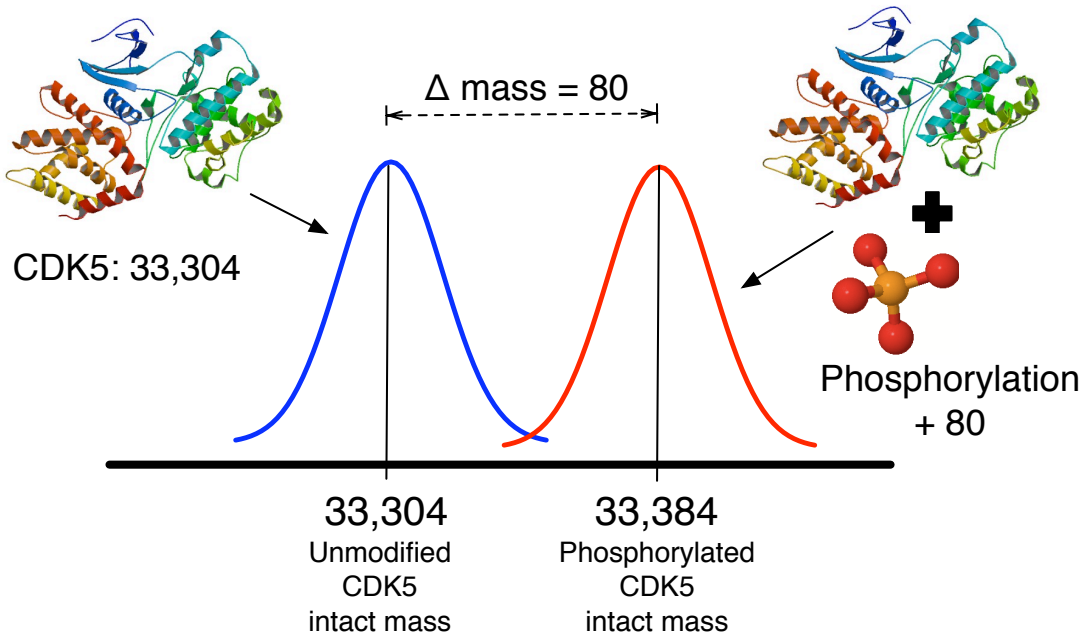
Enhancing the performance of priors by collecting more specific data on terminal

modifications is one possible way to improve the performance of the PIE in this kind of situation, and PIE's data scoring is modular to make just this kind of thing easy. However priors will only ever help fill small gaps in knowledge. It is always the role of the experimental data to do the heavy lifting.

7. I am done!

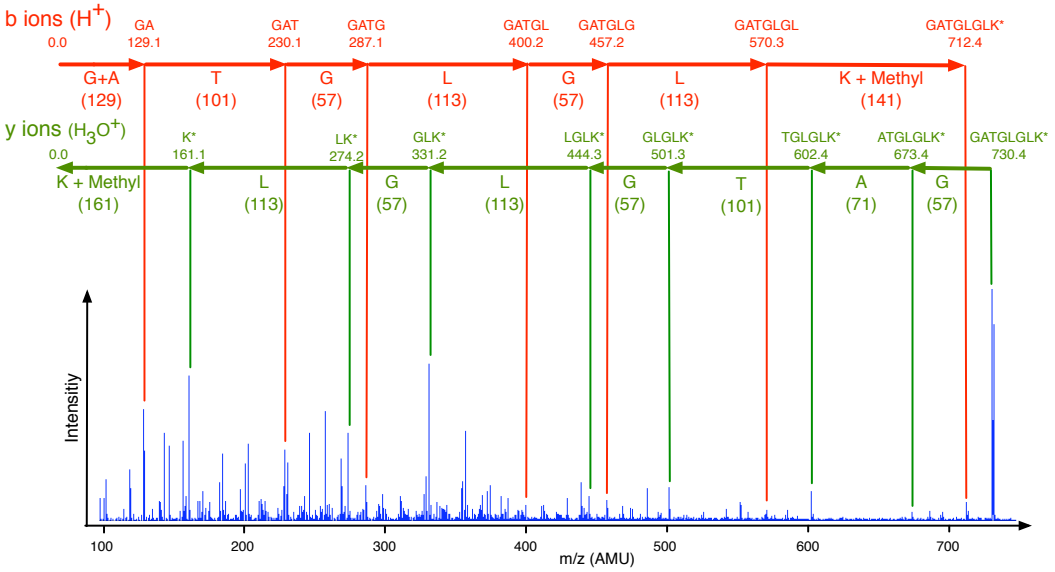
I now have a characterization of the post-translational modification pattern for this protein. If the data had been clean and complete, I would have had a clean and complete answer. Here my data was good but not sufficient to determine a single answer; I have a good idea of what additional data will help the most, and I can go look for evidence of methylation or amidation. After I get some new evidence, I can add it to the PIE and try again.

Figure 1: Intact Mass Shifts Due to PTMs



Modification of a protein will influence its measured mass value. Here the mass of an unmodified (CDK5) protein sequence is 33,304 Da. The mass of the modified protein is 33,384 Da. The difference, +80 Da, is easily discernible by many MS approaches as the difference expected due to a phosphorylation modification. However, it could be due to some other combination of modifications that add up to 80 Da.

Figure 2: Tandem Mass Spectrometry (MS/MS) Locates PTMs

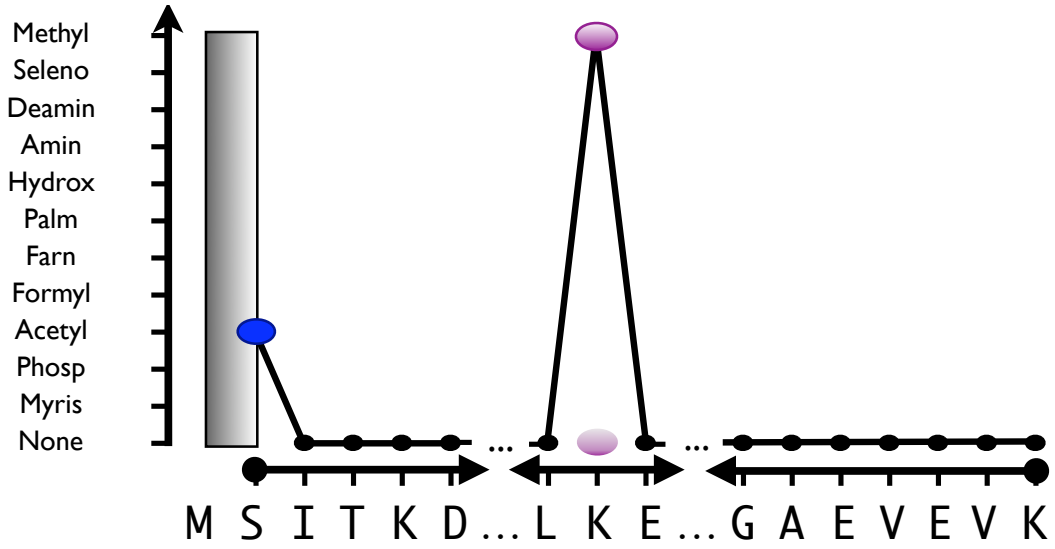


A representative MS/MS spectra for a peptide is shown. Ideally a single random peptide-bond is broken in each peptide molecule. This gives two collection of fragments, one of pieces from the n-terminus side of the break, the other from the C-terminus side. Fragments of all possible substring lengths are generated: N-terminus fragments here are V-, VK-, VKD*-, VKD*L-, etc; c-terminus fragments are R-, RV-, RVG-, RVGP-, etc. Each fragment has a (different) mass, and together result in a sequence of peaks in the spectra. The resulting ladders of mass peaks (one from each end of the peptide) match to increasingly long fragments: 175.12 = R, 274.19 = R+V, 331.21 = R+V+G, etc.

As with intact masses, shifts caused by a modification will be detected. Here a β-methylthiolation modification has occurred on the aspartic acid residue 3rd from the N-terminal end. Instead of the expected mass of V+K+D, at 343, a peak at mass 389 was found. This represents VKD + 46, a mass shift which is attributed to the presence of a β-methylthiolation modification on the D residue.

Unfortunately, interpretation of MS/MS spectra gets complicated. Peptides may be broken in more than one place and at bonds besides those between amino acids, producing distinct ion types. This, along with contaminants or heterogeneous spectra, often results in extra peaks–those not labeled in the diagram. Some expected fragmentation sites may not be seen at all, resulting in a single step of the sequence ladder spanning two or more amino acids, such as occurs for VK in this example. If there are many modifications or the specific sequence of the peptide is unknown, it can be difficult to interpret this spectra, especially since modifications can add to the variability in fragmentation patterns.

Figure 3: Solution Space Representation

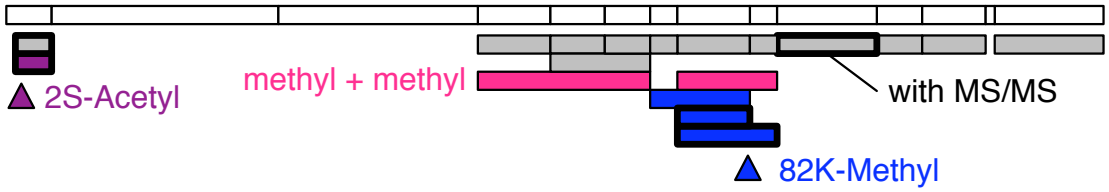


Every possible modified protein that PIE can propose as an answer can be visualized as a jagged line from left to right on the graph shown. The horizontal axis is the canonical protein sequence of the target investigated, here an abbreviated piece of the L7/L12 ribosomal protein. The vertical axis is the set of modifications searched for; the default 11 modifications are shown. Only answers constructed with modifications from this set will be proposed by PIE. Additional or different modifications can be used, with a drop in speed as the number increases. A different, unique line represents each possible modification scenario, formed by connecting the points defined by each adduct modification on the protein. In the example shown, most adducts are none (black points), but there is a (blue) n-terminal acetylation and a (purple) internal K-methylation. To allow for cleavages, the left and right ends of the line do not have to be a the first or the last AA of the protein, An n-terminal methionine truncation is shown as the line starts on the 2nd AA, an S (emphasized by a vertical gray block). This provides a simple way to visualize the multidimensional solution space where points are answers. Points that are close together in the solution space are represented here as segmented lines that are similar and represent similar answers.

Figure 4: H23C Theoretical Peptide Data

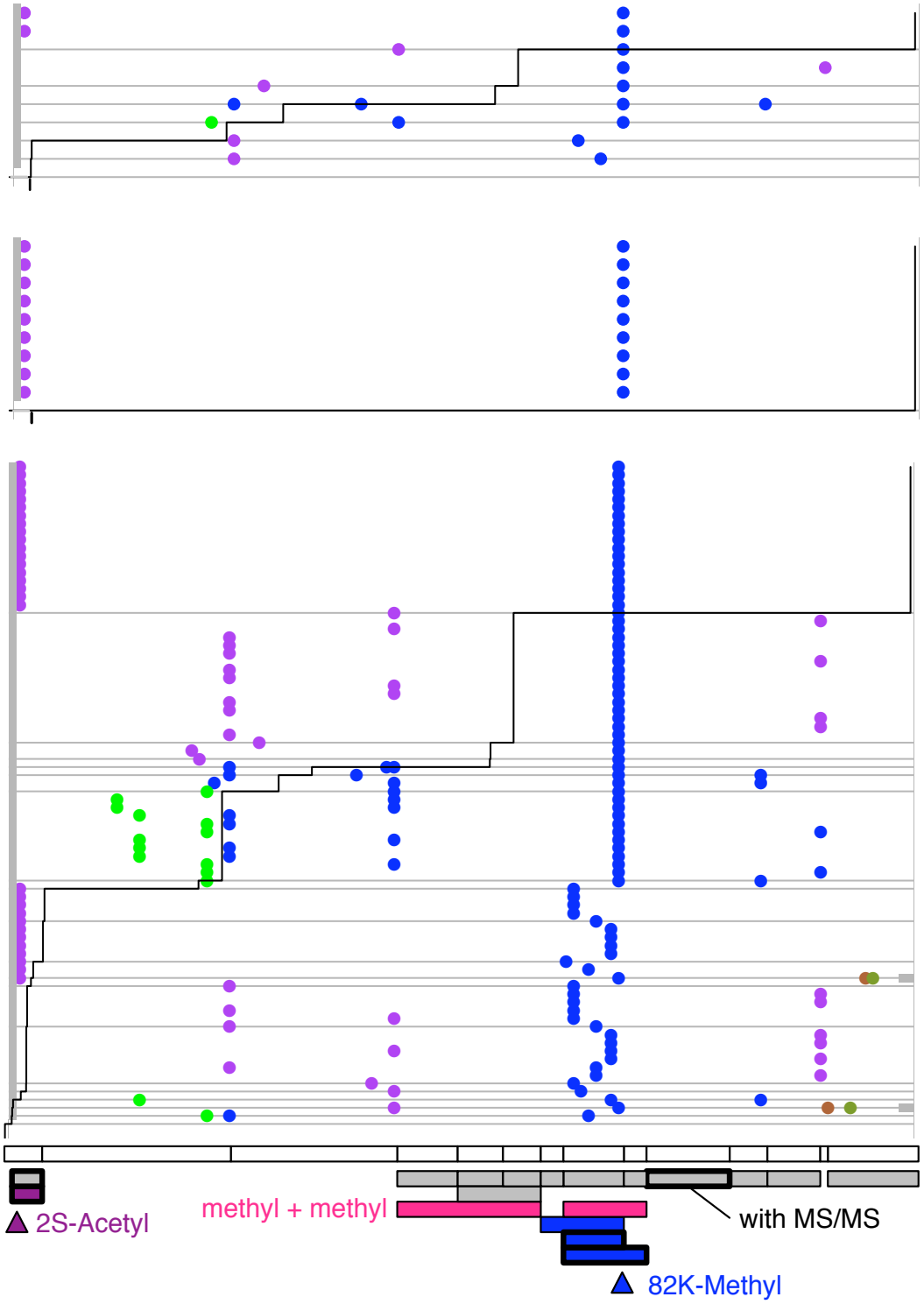
The top white bar shows the theoretical peptide digest of H23C, blocks aligned below indicate theoretical peptides provided to the PIE. Those with thick edges indicate MS/MS sequence data was also provided. Grey boxes indicate unmodified peptides, other boxes are colored based on the modifications as shown. Some regions of the protein have no aligned peptide blocks, simulating missing peptides in the bottom-up data. The second row of peptides are labeled (+H7) to indicate in the H7 bottom-up data set they replace the matching unmodified peptides from the H5 bottom-up data set. Both modified and unmodified peptides are present in the combined bottom-up data sets.

Figure 5: E. Coli L7/L12 Peptide Data



As in Fig. 4, "*H23C Theoretical Peptide Data*", the top white bar shows the theoretical peptide digest, blocks aligned below indicate theoretical peptides. Those with thick edges indicate MS/MS sequence data was provided. Grey boxes indicate unmodified peptides; other boxes are colored based on the modifications as shown. All peptides were provided to PIE, peptides are on separate lines only to show the overlapping and contradictory nature of the data.

Figure 6: Convergence and Profile Sampling, L7/L12 220-1H



Each row represents a different guess proposed by PIE, with guesses in each figure ordered by score from top (hi) to bottom (lo). The peptide data set is reproduced to scale at the bottom of the figure. Colored dots represent modifications–methyl (blue), acetyl (purple), formyl (green), hydroxyl (brown), and myristoyl (olive)–each aligned at their proposed positions. Grey blocks at the left and right indicate n- or c- terminal truncations. Guesses without horizontal grey lines indicate they have the exact same score as the guess above, generating blocks of identically scoring guesses. The jagged black line running roughly diagonally through the graph indicates the score for each row, like a bar graph, except turned on its side with maximum value to the right. Scanning the images by sliding a straight edge top to bottom or left to right provides an "animated" display that helps interpret modification alignments.

Figure 6a. 10 Samples Taken After 50,000 Steps.

Although the highest score (guess) has been found twice, this is not clear evidence for convergence, so a longer length run set is necessary.

FIgure 6b. 10 Samples Taken After 250,000 Steps.

The same highest score (guess) is found here as was found in the 50,000 step set, and It is found many more times. This indicates convergence has probably been reached, and this is likely the best answer given the data.

FIgure 6c. Profile of 100 Samples.

To look for alternate interpretations for the data, a profiles of guesses is sampled at a step length providing 20% of the answers as the best scoring candidate, and 80% as other sub-optimal guesses. Estimated from the convergence plots, Figure 6A shows this to be 50,000 steps. By examining these nearby answers it is possible to see not just how strongly a guess is supported, but how strongly different features of a guess are supported. The best guess with its specifically localized acetylation and methylation is approximately twice as good as the next bess guess (black scoring bar drops to half between them). All high scoring guesses have a predicted n-terminal methionine loss and an 82K methylation, indicating no consistent predictions could be made that didnt include these features. Most sub-optimal guesses have an acetylation. In the best scoring guess its localization is on the new n-terminus, 2S. If not there, the second best scoring guess places it on a lysine aligned with a missing peptide. Here the effect of the prior data module implementing the AA preference of modifications can be seen, as lysine is the most commonly acetylated modification, as well as the effect of the peptide data model, since peptides provide information that any covered lysine is probably not modified. Unfortunately, no other information is available to distinguish between these three positions, so there are three different equally likely predictions made as the second best guess. Other less likely answers include different positions for the acetyl modification, a tetra-methyl species, and a dimethyl + formyl species. Two rare, bad guesses suggest methyl + acetyl + hydroxyl + myristoyl, along with a cleavage of 1 n-term and 2 c-term amino acids, but even these rare guesses are in agreement with the intact mass data. They are approximately isobaric to the best guess, but score much lower due to multiple conflicts with peptide, MS/MS, and prior data modules.

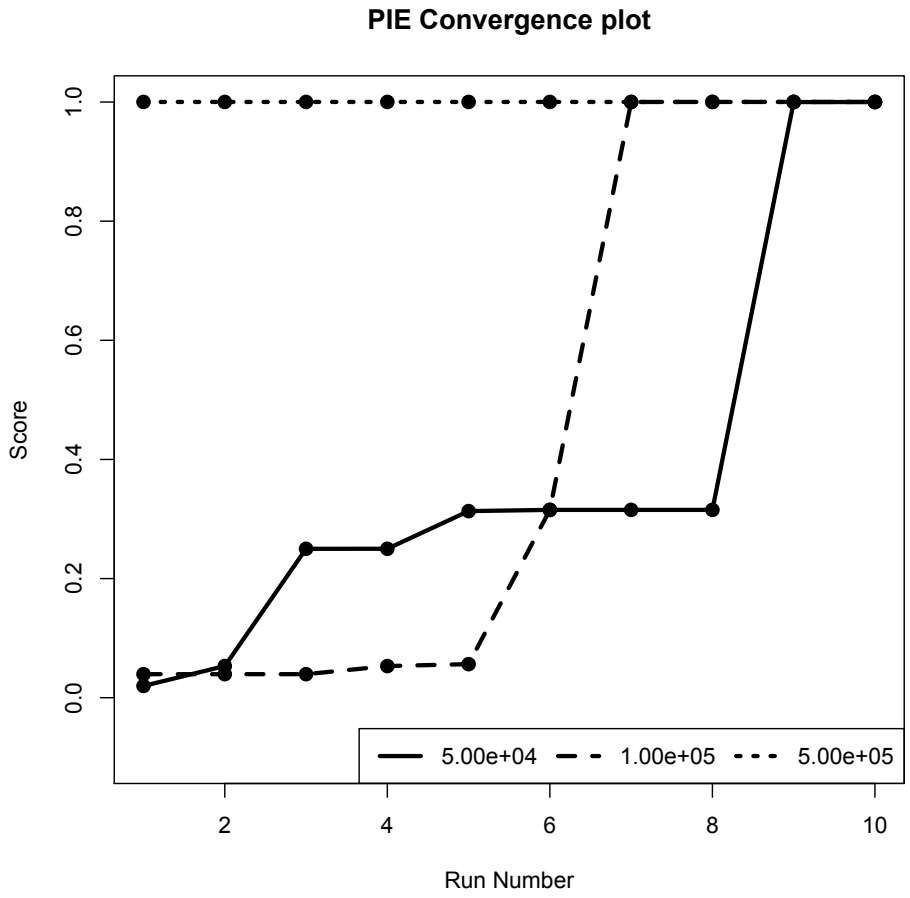
Figure 7: L16 Theoretical Data

An artificial L16-A protein isoform is shown in green, with three adduct modifications represented by triangles: Acetylation at 2-L (blue), Methylation at 49-R (purple), and Oxidation at 107-P (red). There is also a 1 AA truncation at the n-terminus (grey block). This mock target is the answer that the PIE is seeking from the data.

The two sets of bars above the target represent synthetic fragment (grey) and MS/MS (black) data. The MS/MS data identifies AA 107-P as having an oxidation modification (red line), and many others that are unmodified. The peptide data shows no modifications, but does identify additional unmodified regions of the protein. Multiple overlapping peptides with different match scores are represented; the darker the grey, the better the match. Confidence increases (darker grey) when peptides overlap.

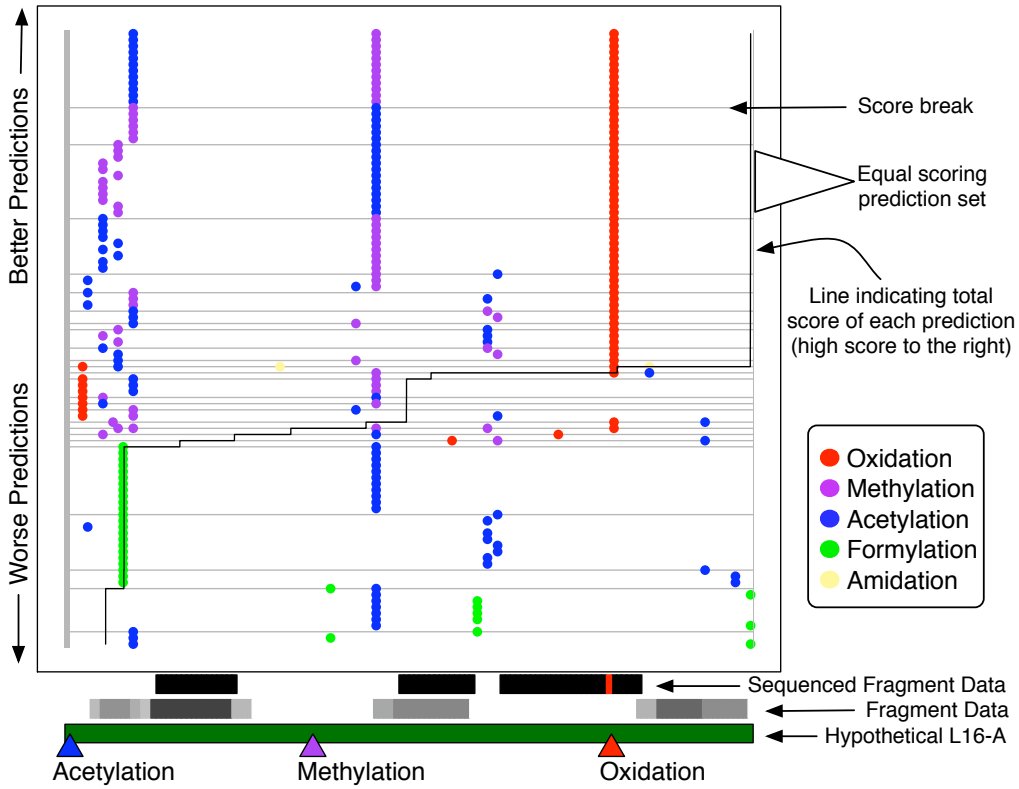
Although providing moderate coverage, the bottom-up data is significantly incomplete. It lacks any indication of the acetylation or methylation modifications, and allows for cleavage of up to 6 n-terminal AA before contradicting any peptide data. Using such incomplete data allows showing both how PIE infers adduct modifications and terminal cleavages from an intact mass, and how incomplete data sets can support multiple answers.

Figure 8: L16 Convergence



Three separate runs were performed for the L16 protein target shown in Fig. 7, "*L16 Theoretical Data*", each consisting of 10 searches but using different search lengths: 50,000 steps (solid) , 100,000 steps (dashed), and 500,000 steps (dotted). For each run, the 10 search results are ordered from lowest scoring to highest scoring and plotted on the graph from left to right. The score axis has been normalized to the largest value found across all runs. The initial 100k step run appears to show convergence, with 4 of the 10 runs having the same high score. As with any other stochastic process (such as measurement!), the results from the PIE will differ in specifics even for identical input, but on average are consistent. The second run at 500k steps, presents only the top scoring candidate, 10 out of 10 times. Together these provide strong evidence that this top scoring candidate really is the best candidate that can be found. A third run of 50K steps gets 2 out of 10 values converging to the same high scoring value, making 50K the approximate optimal convergence length.

Figure 9: L16 Answer Profile



This answer profile represents the combined results from 100 searches, each 100k steps long, for the L16-A target described in Fig. 7, "*L16 Theoretical Data*". The 100 candidate results are stacked vertically, ordered from the lowest scoring (bottom) to highest scoring (top). The horizontal axis represents the amino acid sequence of the protein from n-terminus (left) to c-terminus (right), with only the adduct modifications shown (colored dots positioned where predicted). Consensus modification positions are easily seen as overlapping vertical columns of dots. The left side of the graph shows a grey bar indicating the n-terminal truncation, predicted for every candidate. The relative score of each candidate is indicated by where it crosses the dark-grey score line. This line begins vertically on the right (at 100% relative score), transitions to a jagged horizontal line across the middle, and then ends at the bottom left (about 5% relative score). The horizontal light-grey lines delineate answer sets within which all answers have identical scores.

Most of the high-scoring sets have correctly identified the three adducts and the n-terminal truncation, although there is one high scoring answer (about 85%) that suggests there are two amidations and an extra oxidation instead of the methylation, a surprising answer that can not be ruled out given the available data.

The position of the oxidation modification has also been correctly identified throughout most high scoring candidates, aligning with the oxidation modification presented in the sequenced fragment data. The positions for the methylation and acetylation modification are not correctly aligned with the target, but are generally placed where they do not conflict with the unmodified position data from the MS/MS and peptide data. More data is needed to pin down the exact location of these modifications.

Table 1: Summary of Scoring Functions

Term	Model	Data Type	Model Type
S_1	Intact Mass	Experimental	$1/x$
S_2	Peptide Mass	Experimental	$1/x$
S_3	MS/MS Sequence	Experimental	a_n
S_4	Adduct Frequency	Prior	$\prod_i f_i$
S_5	Adduct Location	Prior	$\prod_i f_i$
S_6	Adduct Count	Prior	$1/x$
S_7	N-Cleavage	Prior	a_n
S_8	C-Cleavage	Prior	a_n
S_9	Rules	Prior	$\prod_i f_i$

Table 2: Analysis of L7/L12 Theoretical Data

Set	Name	Intact (ppm)[a]	Error (ppm)[b]	Peptides Detected[c]	Coverage pep, seq[d]	Steps	Best and 2nd Best Answers[e]	length	Score Ratio	Why?[f]
1	ideal	+ 0.5	1	**1-14**	100%, 100%	50k	Acet-2S, Meth-82K Acet-2S, Meth-**76A**	2–121 2–121	5.72	ms-ms
2	good	-23	20	**1**, 3, **5**, **8, 12**, 14	50%, 25%	25k	Acet-2S, Meth-82K Acet-2S, Meth-**76A**	2–121 2–121	5.72	ms-ms
3	min	+40	50	**1, 8**	10%, 10%	25k	Acet-2S, Meth-82K Acet-**1M**, 2S, **16S**, Meth-82K, **86K**	2–121 **1–119**	5.33	intact
4	no tandem	+40	50	1, 8	10%, 10%	60k	Acet-2S, Meth-82K Acet-2S, Meth-**76A**	2–121 2–121	1.43	mod aa
5	no acetyl	+40	50	**8**	5%, 5%	60k	Acet-2S, Meth-82K Acet-**X**, Meth-82K**, K, K, K**	2–121 2–121	1.16	mod cnt
6	no methyl	+40	50	**1**	5%, 5%	15k	Acet-2S, Meth-K Acet-2S, Meth-**E**	2–121 2–121	1.28	mod aa
7	high intact	+75	100	**1, 8**	10%, 10%	50k	Acet-2S, Meth-82K Acet-2S, Meth-**76A**	2–121 2–121	1.43	mod aa
8	low intact	-90	100	**1, 8**	10%, 10%	35k	Acet-2S, Meth-82K Acet-2S, Meth-**76A**	2–121 2–121	1.43	mod aa
9	no mod	-23	20	3, **4, 5**, **11, 12**, 14	50%, 25%	25k	Acet-2S, Meth-K Acet-2S, Meth-76**A**	2–121 2–121	1.28	mod aa
10	intact	-23	20	n/a	n/a	250k	Acet-2S, Meth-K Acet-2S, Meth-**39A**	2–121 2–121	1.43	mod aa
11	hi intact	+75	100	n/a	n/a	350k	Acet-2S, Meth-**X, Oxid-P, Amid-A** Acet-2S, Meth-K, **Deam-65N**	2–121 2–121	1.04	mod type
12	low intact	-90	100	n/a	n/a	50k	Acet-2S, Meth-K, **Amid-A** Acet-**K**, Meth-K, **Amid-A**	2–121 2–121	1.91	rules

[a] The simulated intact experimental mass, relative to the theoretical intact mass.
[b] The simulated instrument error window, mass measurement are "accurate" with ± this value.
[c] Modified Peptides: 1 has an n-terminal methionine loss and an n-terminal acetlyation at 2S, 8 has a lysine methylation at 82K. Bold peptides have sequence data. [d] Pep. coverage means has matched peptide to region. Seq. coverage means has MS/MS sequence for matched peptide. [e] Top answer scores best. Bold means differs from expected. Acet = acetylation, Met (methylation), Oxid (oxidation), Amid (amidation), Deam (deamidation). Modifications without position represent multiple equal scoring answers that placing a modification at different positions. [f] The scoring module contributing the most to differentiating the top two answers.

Table 3: Analysis of H23C Theoretical Data

Set	Intact (ppm)[a]	Error (ppm)[b]	Peptide Set	Steps	Top answer and 2nd Best Answers[c]	length	Score Ratio	Why?[d]
H5	-10	20	H5 set	250	Acet-5K, 15K, Meth-10K, 24K, Phos-11S Acet-**2**A, 15K, Meth-10K, 24K, Phos-11S	2-136 2-136	2.08	Peptide
H7	-10	20	H7 set	375	Acet-5K, 15K, Meth-10K, 24K, Phos-11S, 12T, 29S Acet-5K, 15K, Meth-10K, 24K, Phos-11S, 12T, **32**T	2-136 2-136	4.35	Peptide
H5-both	-10	20	H5 + H7	75	Acet-5K, 15K, Meth-10K, 24K, Phos-11S Acet-**2**A, 15K, Meth-10K, 24K, Phos-11S	2-136 2-136	2.08	Peptide
H7-both	-10	20	H5 + H7	300	Acet-5K, 15K, Meth-10K, 24K, Phos-11S, 29S, **S** Acet-5K, 15K, Meth-10K, **K, K, K**, Phos-11S	2-136 **1**-136	1.89	Intact

[a] The simulated intact experimental mass, relative to the theoretical intact mass. [b] The simulated instrument error window, mass measurement are "accurate" with ± this value. [c] Top answer scores best. Bold means differs from expected. Acet = acetylation, Met (methylation), Oxid (oxidation), Amid (amidation), Deam (deamidation). Modifications without position represent multiple equal scoring answers that placing a modification at different positions.
[d] The scoring module contributing the most to differentiating the top two answers.

Table 4: Analysis of L7/L12 Experimental Data

Set	Intact (ppm)[a]	Error (ppm)[b]	Manual Interpretation[d]	Steps	Best Answer and 2nd Best Answer [d]	Intact difference (ppm)[e]	Score Ratio	Why?[f]
220-2H	12220.3	50	Acet-2S, Meth-82K	50k	Acet-2S, Meth-82K Acet-**K**, Meth-82K	-17	1.78	Rule
220-1H	12220.1	50	Acet-2S, Meth-82K	50k	Acet-2S, Meth-82K Acet-**K**, Meth-82K	-1	1.78	Rule
207-1L	12206.9	150	Meth-82K, K, K	50k	Meth-82K, K, K Meth-82K, **101**K, K	-65	1.12	Peptide
207-2H	12206.8	50	Meth-82K, K, K	50k	Meth-82K, K, K Meth-82K, **101**K, K	-57	1.12	Peptide
207-1H	12206.5	50	Meth-82K, K, K	50k	Meth-82K, K, K Meth-82K, **101**K, K	-32	1.12	Peptide
206-0H	12206.1	50	Meth-82K, K, K	50k	Meth-82K, K, K Meth-82K, **101**K, K	+0.5	1.12	Peptide
205-0L	12205.5	150	Meth-82K, K, K	50k	Meth-82K, K, K Meth-82K, **101**K, K	+50	1.12	Peptide
175-1M	12174.5	100	Meth-82K	50k	Meth-82K, **K, K** **X-120V, 121K, Amid-55F,** Meth-82K, **118M Myr-115A**	+292	8.96	Peptide
163-1M	12162.9	100		50k	**Meth-82K** **X-2S, Meth-81K,** **Acet-11E, 115A, 116E**	+1246	5.56	MS/MS

[a] The measured intact experimental mass. [b] The estimated instrument error window, mass measurement are assumed accurate with ± this value. [c] Manual interpretation taken from investigation of modification isoforms of all ribosomal proteins. (Ramkisoon, unpublished data).
[d] Top answer scores best. Bold means differs from expected. X (truncation), Acet (acetylation), Met (methylation), Amid (amidation), Myr (myrstolation). All included X-1M, loss of leading methionine. Modifications without position represent multiple equal scoring with given modification at different positions. [e] Difference between intact mass of manual guess and top PIE guess. [f] The scoring module contributing the most to differentiating the top two answers.

Bibliography

Albrethsen, J. (2007). Reproducibility in protein profiling by MALDI-TOF mass spectrometry. Clin. Chem. 53, 852-858.

Banerjee, A. and Gerondakis, S. (2007) Coordinating TLR-activated signaling pathways in cells of the immune system. Immunol Cell Biol, 85, 420-4.

Bendtsen, J. D. et al. (2004) Improved prediction of signal peptides: SignalP 3.0. J Mol Biol, 340, 783-95.

Blom, N. et al. (1999) Sequence and structure-based prediction of eukaryotic protein phosphorylation sites. J Mol Biol, 294, 1351-62.

Bogdanov, B. and Smith, R. D. (2005) Proteomics by FTICR mass spectrometry: top down and bottom up. Mass Spectrom Rev, 24, 168-200.

Bolte, M. and Hogan, C. J. (2002) Conflict over the age of the Universe. Nature, 376, 399-402.

Brocchieri, L., and Karlin, S. (2005). Protein length in eukaryotic and prokaryotic proteomes. Nucleic Acids Res. 33, 3390-3400.

Craig, R. and Beavis, R. C. (2004) TANDEM: matching proteins with tandem mass spectra. Bioinformatics, 20, 1466-7.

Creasy, D. M. and Cottrell, J. S. (2004) Unimod: Protein modifications for mass spectrometry. Proteomics, 4, 1534-6.

Domon, B. and Aebersold, R. (2006) Mass spectrometry and protein analysis. Science, 312, 212-7.

Durbin, K. R. et al. (2010) Intact mass detection, interpretation, and visualization to automate Top-Down proteomics on a large scale. Proteomics, (in press).

Eng, J. K., McCormack, A. L., and Yates III, J. R. (1994). An approach to correlate tandem mass spectra data of peptides with amino acid sequences in a protein database. J. Am. Soc. Mass Spectrom. 5, 976-989.

Fang, Y. et al. (2010) Quantitative analysis of proteome coverage and recovery rates for upstream fractionation methods in proteomics. J Proteome Res, 9, 1902-12.

Frottin, F., et al. (2006). The proteomics of N-terminal methionine cleavage. Molecular Cell Proteomics. 51, 2336-2349.

Gasteiger, E. et al. (2005) Protein identification and analysis tools on the ExPASy server. In Walker, J. M.(ed), The proteomics protocols handbook. Springer, Heidelberg, Ger., 571–607.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. J. Mol. Biol. 162, 705-708.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. Biometrika. 57, 97-109.

Holmes, M. R. and Giddings, M. C. (2004) Prediction of posttranslational modifications using intact-protein mass spectrometric data. Anal Chem, 76, 276-82.

Huelsenbeck, J. P., Ronquist, F., Nielsen, R., and Bollback, J. P. (2001). Bayesian inference of phylogeny and its impact on evolutionary biology. Science. 294, 2310-2314.

Jefferys, S. R. and Giddings, M. C. (2011) Automated data integration and determination of posttranslational modications with the protein inference engine. Wu, C.H. and Chen, C. (eds) In Bioinformatics for Comparative Proteomics. Springer. Heidelberg, Ger.

Jefferys, S. R. and Giddings, M. C. (2011) Baking a mass-spectrometry data PIE with McMC and simulated annealing: predicting protein post-translational modifications from integrated top-down and bottom-up data. Bioinformatics. 27, 844-852.

Kelleher, N. L., et al T. (1999). Localization of labile posttranslational modifications by electron capture dissociation: the case of gamma-carboxyglutamic acid. Anal. Chem. 71, 4250-4253.

Kentner, D., and Sourjik, V. (2006). Spatial organization of the bacterial chemotaxis system. Curr. Opin. Microbiol. 9 619-624.

Kertesz, V. et al. (2009) PTMSearchPlus: software tool for automated protein identification and post-translational modification characterization by integrating accurate intact protein mass and bottom-up mass spectrometric data searches. Anal Chem, 81, 8387-95.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science. 220, 671-680.

Kollmann, M., et al (2005). Design principles of a bacterial signaling network. Nature. 438, 504-507.

Kysely, J. (2009) Coverage probability of bootstrap confidence intervals in heavy-tailed frequency models, with application to precipitation data. Theor Appl Climatol, 101, 345-361.

Lee, T. Y., et al. (2006). dbPTM: an information repository of protein post-translational modification. Nucleic Acids Res. 34, D622-D627.

Little, D. P., et al. (1994). Infrared multiphoton dissociation of large multiply charged ions for biomolecule sequencing. Anal. Chem. 66, 2809-2815.

Mann, M. and Jensen, O. N. (2003). Proteomic analysis of post-translational modifications. Nat. Biotechnol. 21, 255-261.

Maxwell, E. J. and Chen, D. D. Y. (2008) Twenty years of interface development for capillary electrophoresis-electrospray ionization-mass spectrometry. Anal Chim Acta, 627, 25-33.

Metropolis, N. et al. (1953) Equation of state calculations by fast computing machines. J Chem Phys, 21, 1087-92.

Minamoto, T. et al. (2001) Distinct pattern of p53 phosphorylation in human tumors. Oncogene, 20, 3341-7.

Mirzaei, H. and Regnier, F. (2006) Enhancing electrospray ionization efficiency of peptides by derivatization. Anal Chem, 78, 4175-83.

Monigatti, F., Gasteiger, E., Bairoch, A., and Jung, E. (2002). The Sulfinator: predicting tyrosine sulfation sites in protein sequences. Bioinformatics. 18, 769-770.

Perkins, D. N. et al. (1999) Probability-based protein identification by searching sequence databases using mass spectrometry data. Electrophoresis, 20, 3551-67.

Perry, R. H. et al. (2008) Orbitrap mass spectrometry: instrumentation, ion motion and applications. Mass Spectrom Rev, 27, 661-99.

Plato. (1968) The Republic of Plato.Basic Books, New York, NY.

Ramkisson, K., Su, H., Hamlet, E., Giddings, M. C. (Unpublished) Data from an investigation of E. coli ribosomal proteins and their PTMs by top-down and bottom-up MS.

Searle, B. C, Dasari, S., Wilmarth, P. A., et l (2005). Identification of protein modifications using MS/MS de novo sequencing and the OpenSea alignment algorithm. J. Proteome Res. 4, 546-554.

Seo, J. and Lee, K. J. (2004). Post-translational modifications and their biological functions: proteomic analysis and systematic approaches. J. Biochem. Mol. Biol. 37, 35-44.

Shi, Y. (2007) Histone lysine demethylases: emerging roles in development, physiology and disease. Nat Rev Genet, 8, 829-33.

Siuti, N., and Kelleher, N. L. (2007). Decoding protein modifications using top-down mass spectrometry. Nat. Methods. 4, 817-821.

Strader, M. B., Verberkmoes, N. C., Tabb, et al. (2004). Characterization of the 70S Ribosome from Rhodopseudomonas palustris using an integrated "top-down" and "bottom-up" mass spectrometric approach. J. Proteome Res. 3, 965-978.

Tsur, D. et al. (2005) Identication of post-translational modications via blind search of mass-spectra. Proc IEEE Comput Syst Bioinform Conf, 157-66.

VerBerkmoes, N. C., Bundy, J. L., Hauser, L., et al. (2002). Integrating 'top-down" and "bottom-up" mass spectrometric approaches for proteomic analysis of Shewanella oneidensis. J. Proteome. Res. 1, 239-252.

Wilkins, M. R., Gasteiger, E., Gooley, A. A., et al. (1999). High-throughput mass spectrometric discovery of protein post- translational modifications. J. Mol. Biol. 289, 645-657.

Walsh, C. T., Garneau-Tsodikova, S., and Gatto, G. J. (2005). Protein posttranslational modifications: the chemistry of proteome diversifications. Angew. Chem. Int. Ed. Engl. 44, 7342-7372.

Wisz, M. S, Suarez, M. K, Holmes, M. R, and Giddings, M. C. (2004). GFSWeb: a web tool for genome-based identification of proteins from mass spectrometric samples. J. Proteome Res. 3, 1292-1295.

Yates, 3rd, J. R. et al. (1995) Method to correlate tandem mass spectra of modified peptides to amino acid sequences in the protein database. Anal Chem, 67, 1426-36.

Yates, 3rd , J. R. et al. (2009) Proteomics by mass spectrometry: approaches, advances, and applications. Annu Rev Biomed Eng, 11, 49-79.

Yu, Y., Ji, H., Doudna, J. A., and Leary, J. A. (2005). Mass spectrometric analysis of the human 40S ribosomal subunit: native and HCV IRES-bound complexes. Protein Sci. 14, 1438-1446.

Zhang, H. (2004) The Optimality of Naive Bayes. In Barr, V. and Markov, Z.(ed), Proc. 17th Internat. FLAIRS Conf.. Florida AI Research Society, FL, USA.

Zubarev, R. A., Haselmann, K. F., Budnik, B., et al (2002). Towards and understanding of the mechanisms of electron-capture dissociation: a historical perspective and modern ideas. Eur. J.. Mass. Spectrom. 8, 337-349.