University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Pep2Path v2

Ross McBride

Abstract

Education Use Consent

Contents

1 Introduction

One particular problem of interest for biologists is the identification of new biological products with novel properties. Until relatively recently, this process was done manually by trained experts. But with the recent advent of the field of bioinformatics, we can now bring immense computational resources to bear on problems like this, saving precious expert time.

Pep2Path (Medema et al. 2014) is a tool that aims to accelerate drug identification by matching mass-spectrometry outputs to identified biosynthetic gene clusters and in the original paper the authors demonstrate robust benchmarking results. However, despite the utility of Pep2Path, it is implemented as a few monolithic scripts, with little room to revise elements of it in accordance with new developments and research without a complete rewrite. In this paper we present a small suite of software tools implementing a platform for functionality like Pep2Path, and the software process by which this software was designed and created.

1.1 Aims

The aims of the project are to:

- Implement a set of software tools that can achieve similar functionality to the original Pep2Path, subject to correctness testing.
- Achieve a degree of flexibility in the implementation such that this software can be responsively updated to changes in its environment.
- Follow good software practises in the implementation such as loose coupling such that we can achieve software reuse, a cornerstone of software development.

1.2 Motivation

Various classes of biological compounds are formed from assembly-chains of smaller compounds, such as *peptides*, which are formed from amino acids. If we look at this problem as a computer scientist might, the first thing to come to mind in identifying useful drugs within this space might be an exhaustive search of all combinations of molecule. (Of course, for some molecules questions of topology and structure come into question...) This quite quickly runs into problems, as with the addition of each component to a chain of biological molecules, the chemical space grows exponentially. For example, if we consider the relatively limited case of the twenty *proteinogenic* (those involved in the synthesis of proteins, essential for organic life) amino acids and a peptide of length 4, there are 20^4 possible combinations... Around 160,000! We would then have to identify which, if any, of these have useful properties.

Without some heuristic (for example being able to infer properties from chemical composition), an exhaustive search of the chemical space is impossible. Instead we identify biological products produced in the wild, by living organisms - so-called *'natural products'*, which have seen a recent resurgence in study. (A.Khan 2018) From an evolutionary perspective, we expect that if organisms are expending resources on producing a particular biological product, then it confers a survival advantage. For example, a strain of bacteria might produce an antibiotic that kills another

competitor species of bacteria - which we can then use ourselves to kill a potentially deadly strain of bacteria. Natural products produced in this way to fulfil an auxiliary function not directly related to the growth or reproduction of the organism are known as *'secondary metabolites'.* (Breitling et al. 2013)

To identify potentially interesting natural products we could, for example, analyse chemical structure for similarity to other interesting compounds, or by observing similar behaviour across several species which produce a particular product. However, we also want to match this natural product to the gene cluster - a Biosynthetic Gene Cluster, henceforth **BGC** - which encodes it. One way of achieving this is to cluster together species that share a BGC and produce the natural product in question - this is a good indication that the pattern is meaningful and the BGC produces the natural product.

Mass spectrometry allows us to identify the chemical composition of compounds - by breaking up molecules and then measuring the mass lost, we can infer the *potential* composition of a compound by the masses of its components using a mass-translation table. Mass spectrometry readings, or amino acid **sequence tags** (short listings of consecutive compounds), are one of the key inputs to Pep2Path. Although there are many classes of natural product for which mass spectrometry can be used effectively, (Smith et al. 2014) here we focus on two particular classes, for which Pep2Path provides an algorithm each: Non-Ribosomal Peptides (henceforth **NRPs**) and Ribosomally-synthesized and Post-translationally-modified Peptides (henceforth **RiPPs**).

The other key input to each of the two Pep2Path algorithms is the gene sequence we are attempting to match our compound to. As their name suggests, RiPPs are synthesised by *ribosomes*, the cell organelle responsible for translating gene sequences to proteins and then modified post-translation by enzymes. As a result, they are precisely encoded in an organism's gene sequence and and we can retrieve their composition directly, as Pep2Path does, by obtaining their **'six translation frames'** (the six ways the gene sequence can be potentially be read). NRPs, by contrast, are instead synthesised by Non-Ribosomal Peptide Synthetases (henceforth **NRPSes**) which are independent of the ribosome and can introduce non-proteinogenic amino acids. These consist of several NRPS modules which form an assembly-line of amino acids, and are controlled by three domains - *A (Adenylation)*, *T (Thiolation)* and *C (Condensation)*. Of particular interest to us is the *Adenylation* domain, which controls which substrate is added to the chain of amino acids. These are not directly encoded into the gene and are therefore harder to predict, so we rely on an external tool - **antiSMASH** (Blin et al. 2017) - for this purpose.

Taking these two sets of inputs, **NRP2Path** matches the potential chemical compositions generated by mass spectrometry analysis to BGC predictions using antiSMASH, whereas **RiPP2Path** matches these potential chemical compositions to the six-frame translation extracted from the genetic sequence data of the RiPP-producing BGCs. Pep2Path can then automatically score these two sets of data against one another, filling a key part in an automated drug-identification pipeline, where previously investigation of natural products would be based on hand-identification.

However, the original Pep2Path is written almost entirely using singular scripts, one for each algorithm. As a result, it breaches some important software design principles, by mixing concerns such as parsing input and generating sequence tags, and at times relies on data structures passed around the entire script. This design makes Pep2Path difficult to update, but there are a number of reasons why it might be advantageous to do so. For one, the scoring method the authors use is based on antiSMASH 2.0 (Blin et al. 2013) the current version at the time. Since the release of Pep2Path, there have been antiSMASH versions up to 4. (Blin et al. 2017) While the original Pep2Path scoring method is backwards-compatible, antiSMASH version 4.0 offers the **SANDPUMA** ensemble algorithm (Chevrette et al. 2017) which collates the **Stachelhaus Code** and **SVM** predictors that antiSMASH 2.0 uses along with several others to offer a more accurate BGC prediction.

Furthermore, we may want to investigate product classes other than NRPS and RiPP, the classes

of biological product on which Pep2Path operates – there are many others, including *polyketides* and *alkaloids*. (For an example as to why this might be useful, the pain-medication morphine is an alkaloid extracted from the opium poppy.) It would also be useful to integrate different algorithms for identifying biological products from sequence data (such as the **RiPPQuest** (Mohimani et al. 2014) method for *lanthipeptides* – a subtype of RiPP), decouple the mass spectrometry process from the Pep2Path process or otherwise update the scoring mechanism with new developments. We also expect more generally that having a more transparent and modular software design might lead to easier discovery of bugs or integration with other codebases.

2 Background

The motivation for Pep2Path comes from the recent technology of **Peptidogenomics**. (Kersten et al. 2011) In this original paper the authors propose the method of comparing mass spectrometry sequence tags to translated/predicted genomes for ribosomally-synthesised and non-ribosomally synthesised peptides respectively in order to mine for interesting natural products. However, while there were computational tools for aspects of this process, such as the interpretation of mass spectra and the interpretation of genomes, there was no end-to-end tool automating Peptidogenomics - until Pep2Path. During this section we discuss the background behind Peptidogenomics and Pep2Path, and some of the relevant literature.

2.1 Mass Spectrometry

Mass spectrometry is a common technique in chemistry for the identification of the chemical composition of a molecule by ionising it, causing it to break into fragments. There are multiple approaches to mass spectrometry, and multiple ways to interpret the data - for interpretation, there exist dereplicator tools like *iSNAP* (Ibrahim et al. 2012) and *Dereplicator+* (Mohimani et al. 2018) which attempt to statistically match mass spectra output to *known* molecules in a database. Dereplicator+ for example, functions by comparing them to sample mass-spectra generated from a known database of chemicals, by simulating how the molecules of those spectra will fragment.

However, for our purposes we have a series of ordered readings of mass and intensity - gaps in mass peaks correspond to fragmented parts of a molecule, and we can then translate the mass being broken off into a chemical composition using knowledge of molecular weights stored in a dedicated mass translation table. (The so-called *'de novo'* mass spectrometry technique - alternative methods are an active area of research.) Traditionally, this was a problem hand-solved by chemists, but has long been a target for computational processes due to the ease of translation of these numerical processes.

Once mass shifts have been translated to potential compounds, these can be joined together, end-to-end into longer sequences of compounds - 'sequence tags'. Then, computational resources can be used to easily mine a group of mass spectrometry readings for sequence tags. However, there will inevitably be noise in mass spectrometry readings, whether from measurement error or the breaking off of small fragments that don't represent a significant part of the molecule. For this reason, two variables are introduced: an *intensity threshold* and a *mass tolerance*. We can firstly cut out all low-quality readings by cutting out all readings below a certain intensity; secondly, we can account for slight discrepancies in mass measurements by measuring mass shifts within some interval rather than taking the exact values from the mass table. Good values for these variables depend on the dataset in question, and a good general pair of values is currently unknown, but processing mass spectra in this way is standard across most approaches.

2.2 BGCs

2.2.1 NRPs and antiSMASH

antiSMASH (antibiotics and Secondary Metabolite Analysis Shell) (Blin et al. 2017) is a widely-used piece of bioinformatics software used for the automated labelling of BGCs and genome-mining of secondary metabolites from raw sequence genome sequence data extracted from bacteria, fungi or plants. It is usable both as a standalone program on MacOS or Linux and as a web-server and has gone through several revisions and is, at the time of writing, currently up to version 4.0. Therefore, it aggregates many of the latest developments in natural product research and offers many features for the handling of such sequence data and the annotation of its BGCs, outputting data in the standard GenBank format.

For the purposes of this paper, we are interested in antiSMASH's ability to predict the substrate specificity of an NRPS' adenylation domains, that is, give us a prediction for the chemical makeup of a particular NRP given the sequence data of the organism that produces it. The current version of antiSMASH uses the SANDPUMA ensemble algorithm to do this, which aggregates the results of several other predictive algorithms, including those in previous versions of antiSMASH (maintaining backwards-compatibility), to produce significantly better results. However SANDPUMA and antiSMASH 4.0 are relatively recent compared to Pep2Path, and instead we are interested in two key predictors of antiSMASH 2.0: the Stachelhaus Code, a set of rules for comparing different adenylation domains created from empirical observation (Stachelhaus et al. 1999) and a machine learning Support Vector Machine (SVM) based method.

Machine learning is frequently used in bioinformatics to extrapolate underlying trends from the vast quantities of data often involved, learning from a provided dataset useful understandings of programmer-selected features by performing some optimisation task. SVMs in particular are a classifier (that is, assign inputs to one of a set of discrete classes, in this case a substrate prediction) and attempt to draw a decision boundary separating those classes so as to minimise the distance between a certain number of the training datapoints plotted in a hyperplane. The SVM implementation for antiSMASH is provided by **NRPSPredictor2**, (Röttig et al. 2011) a standalone piece of software which has since been integrated into the antiSMASH pipeline. One of the the things that makes the original Pep2Path results robust is that it was tested on datasets NRPSPredictor2 had *not* been trained on, avoiding the introduction of bias and showing the generalisability of the Pep2Path method.

2.2.2 RiPPs, their six-frame translation and RiPPQuest

For RiPPs, the translation process is more direct than with NRPs, and does not require an external platform like antiSMASH to make substrate predictions. This process can be done via the 'six-frame translation'. DNA strands are made of long strings of four *nucleotide* bases *(Adenine, Glycine, Thymine and Cytosine)* which can easily be represented and processed in computer systems in large numbers, and which bind in A-T and G-C pairs across the two strands. These bases, in groups of three known as *codons*, reliably encode amino acids in ways we can extract. However, when looking at a genome, we do not know where the first codon begins - it could begin at any of three positions. Then, the encoding could be done by either strand - we only store one strand, but we can retrieve its *reverse complement* by converting the base to its corresponding base and reversing the strand - for a total of six different encodings. This is the six-frame translation method we use for RiPPs.

Among RiPPs, there are many subtypes. One particular subclass of interest is the lanthipeptide, which is specifically targeted by the bioinformatics software RiPPQuest. (Mohimani et al. 2014) A successor to the original Peptidogenomics paper, the RiPPQuest method centres on the prediction of the *'LANC-like'* domain in the genome, which is important for the biosynthesis

of lanthipeptides in particular. It then centres its translation window around the LANC-like domain and begins searching using the six translation frames.

One particular note is that as sequence data gets larger, so will the probability of random matches to our sequence tag. This method's performance improves as we have either longer sequence tags, or shorter sequence data. If we assume (as a purely illustrative exercise) that there is uniform probability for each sequence tag across the chemical space, for the 20 proteinogenic amino acids (the alphabet for RiPPs) then the probability of any given sequence tag of length 2 is $\frac{1}{400}$. Relatively likely, even in small data. However, a peptide of length 4 would have the probability $\frac{1}{160000}$. Of course, longer genome sequences may contain enough data to appear to have several million peptides and *still* randomly match tags of length 4, 5 or even upwards, but it gets exponentially less likely as sequence tag length increases.

One of the motivations for RiPPQuest is that lanthipeptides have relatively short sequence tags, and thus it is necessary to target the sequence length. By targeting the translation window around a LANC-like domain, RiPPQuest searches less of the genome and provides more accurate results, but consequently only operates on lanthipeptides in particular. (These are some of the features relevant for comparison to our method - there are other complexities to their method which we will not remark on here.) We do not implement the RiPPQuest method here, and instead implement a more general method with looser assumptions in order to target all classes of RiPP, but we highlight this method as a point of comparison and possibility for future extension.

2.3 Pep2Path

Pep2Path (Medema et al. 2014) provides two algorithms, NRP2Path and RiPP2Path. Both rely on the same principles of mass spectrometry. In the original Pep2Path source program, mass search tags can either be given directly or can be derived from a mass-shift sequence. They then extract relevant BGC information from a sequence, and compare potential sequence tags to potential BGCs using a different scoring function for each algorithm in order to enable finding the best match.

2.3.1 NRP2Path

NRP2Path first takes potential sequence tags and antiSMASH substrate specificity predictions. These sequence tags can be arranged either forwards or backwards, and within the prediction there are several different NRPS modules, which themselves can either be arranged forwards or backwards and can be arranged in any order to make up the full BGC sequence. So in order to test all possible gene sequences, Pep2Path must generate all permutations of the cartesian products of the forwards and backwards modules (a total of $n! \cdot 2^n$ different orderings for n modules). A sequence tag and an ordering of a gene sequence are then aligned with one another and scored for their match, testing every possible alignment in order to find the best score.

In order to compare a sequence tag to a BGC, NRP2Path uses its own scoring function loosely inspired by *Bayes' Rule*. We omit the details of its derivation here for brevity's sake, but they are available in the original paper.

$$S(C|T) = \sum_{A \in T} ln\left(\frac{P(A) + c \cdot ({I_{A,M}}^{\eta} + x \cdot P(A))}{P(A) \cdot (1 + c \cdot (\sum_{A \in \mathbb{A}} ({I_{A,M}}^{\eta} + x))}\right)$$

$S(C|T)$ is the score of a gene cluster given the sequence tag. A is the amino acid making up part of a tag, and $\mathbb{A}$ is the total amino acid alphabet being used. c and x are parameters that express

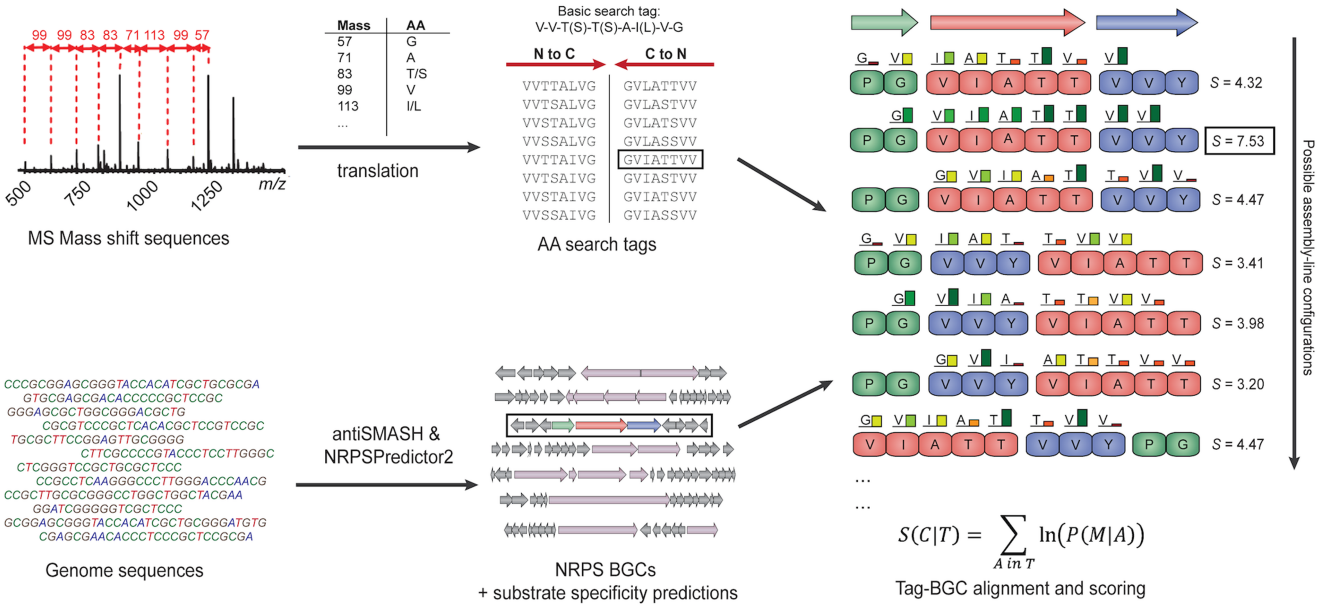

Figure 2.1: *An illustration of the NRP2Path algorithm. On the top is a mass spectrometry output being put through a mass translation table to produce a comprehensive list of potential sequence tags. On the bottom is BGC substrate specificity being extracted from raw sequence data by antiSMASH. These two sets of data are then used to compare every alignment of every sequence tag to every possible ordering of NRP blocks extracted from the BGC. using the abbreviated scoring function shown below. Figure adapted from Medema et al. (2014).*

the degree of confidence that final probability relies on substrate specificity rather than prior probability, and a pseudo-count to correct for small sample size, with default values $c = 1$ and $x = 0.01$, respectively. η is a regularisation term allowing for the exponential penalisation of repeated mismatches.

$I_{A,M}$ is the average of two values calculated separately for the Stachelhaus Code and SVM predictions. The Code prediction is based on the closeness to the closest known NRPS module for NRPSPredictor2, whereas the SVM value is assigned between five evenly-spaced thresholds between 0 and 1 based on how closely the classes of the amino acid match (0 for no relation, 1 for exact match).

$P(A)$ is the prior probability of A, calculated like so:

$$P(A) = \frac{n(A) + \frac{k}{|\mathbb{A}|}}{\sum_{(A \in \mathbb{A}} n(A) + k)}$$

where k is another pseudocount with k=1 and n(A) is the number of appearances within the NORINE database. (Caboche et al. 2008)

Finally, the degree of matching between any gene sequence and mass spectrometry output is defined as the maximum of its scores across any alignment.

2.3.2 RiPP2Path

While RiPPQuest provides a method for the matching of BGCs to mass spectrometry readings in lanthipeptides, it does not cover all RiPPs. Following RiPPQuest, Pep2Path introduced RiPP2Path, a simple accessory tool for more broad matching of RiPPs. It functions by running a sliding window containing the sequence tag over every position of every translation frame in a gene sequence, computing the number of amino acid matches over the total length of the tag and then returning the highest scores.

Unlike RiPPQuest, this algorithm is naively targeted and processes a score for each gene subsequence, distinguishing them only by the score they receive. This does have penalties for accuracy where any identified sequence is more likely to have have appeared purely by random chance, especially in larger data, and costs more processing power inasmuch as it searches more, but it comes with the benefits of a very simple algorithm and is capable of applying the technology of peptidogenomics to all classes of RiPP, not just lanthipeptides.

3 Analysis/Requirements

3.1 Requirements

For this project, the supervisor acted as a client, proposing the software for eventual integration into a larger codebase, and due to greater domain experience provided requirements implicitly as the project progressed.

Firstly, we wanted a set of tools to implement the original Pep2Path functionality. This meant implementing NRP2Path and RiPP2Path. Secondly, the client had their own dataset for which the alignment comparison of NRP2Path might prove particularly cumbersome. This then meant being able to read the formats of the data available, and implement a simpler algorithm comparing the overlap between components. We also desired these implementations to be flexible with respect to future advances in bioinformatics, allowing the integration of custom scoring functions, as we outlined in the introduction. (1.2) We implement these described algorithms both to solve this more proximal problem, and demonstrate the flexibility of our implementation.

The progam has to handle some very large search spaces, so efficiency is a concern. However, the client had a preliminary stage of screening for data reducing the data our implementation would have to process. Additionally, it would also be run on powerful hardware dedicated to processing very large bioinformatics data, so whilst it was necessary for our implementation to be efficient enough to terminate in reasonable time (and not, say, have unnecessary orders of complexity included within algorithms), the absolute highest degree of efficiency was not an aim for this project.

<note to explain some of the other requirements below>

<note on the importance of an end-to-end method - justify why we would want to reinvent the wheel badly>

We list functional requirements here using the MoSCoW method, and then non-functional requirements separately.

3.1.1 Must Have

- A mass spectrometry tool capable of converting mass/intensity readings to sequence tags.
- Various small tools to be able to read in input data from standard file formats. Particularly, the ability to read the domain-standard antiSMASH-produced Genbank file formats.
- A software base allowing the integration of custom scoring functions, and allows the provision of custom mass tables/alphabets for the compounds being operated on.
- Implementation of a simpler BGC/mass-spectrometry comparison equivalent to taking the set intersection of the components in both, more suited to datasets with shorter sequence tags.
- A full implementation of NRP2Path.
- A suite of unit tests verifying the behaviour of the implementations of the various algorithms and improving future maintainability.

3.1.2 Should Have

- A full implementation of RiPP2Path.
- The ability to run a 'many-to-many' comparison between BGC-predictions and mass spectrometry.
- Replications of the original experimental results for Pep2Path (not necessarily exact, but on-target) to demonstrate functional correctness by practical test.
- Integration with the client's codebase.

3.1.3 Could Have

- A simple CLI to run as a standalone.
- An accessory visualisation tool to plot mass spectra with predicted sequence tags over them.
- An adaptation of the RiPPQuest method to narrow down the translation around to just the region around the LANC-like domain for lanthipeptides.

3.1.4 Won't Have

- Implementations of novel scoring functions, such as one that takes advantage of antiSMASH 4's use of the SANDPUMA ensemble.
- A full implementation of the RiPPQuest method for genome mining on lanthipeptides.

3.1.5 Non-Functional Requirements

- Should maintain strong separation of concerns and loose coupling, so components can be reused.
- Platform-independence (portability).
- Transparency of design.
- A degree of efficiency - not a priority, but the program should avoid wasting time where possible.

4 Design

4.1 Software Structure

Examining our proposed software, a natural design for the software emerges. There are three important concerns: mass spectrometry, sequence handling, and the implementation of the two Pep2Path algorithms themselves, reliant on the two prior concerns. Therefore we propose a design to separate our software into corresponding packages.

One module must handle the parsing of any relevant MS data file formats from our client's data, the extraction of sequence tags from such data and a possible accessory tool to plot the sequence tag along its corresponding spectrum. Another must handle the extraction of useful information from sequence data - that is, adenylation domain predictions and raw sequence data from antiSMASH-produced Genbank files, for NRP2Path and RiPP2Path respectively. Finally, the last module must have both of these as dependencies, and implement NRP2Path and RiPP2Path. Additionally, if a standalone CLI were to be implemented, this would then wrap around the outside of the software, treating the entire thing as a dependency.

This design permits us to separate our implementation of Pep2Path itself from the data fed into it, whilst still providing an end-to-end implementation from raw MS data and antiSMASH data to match scoring.

4.2 Algorithms

During this section, we describe precisely the various algorithms we use for this project - note that for MS-dereplication and NRP2Path, we do not specify a mass table/alphabet respectively, and allow the user to provide this. In the case of mass spectra this is totally context-dependent on the molecules being fragmented - in the case of NRP2Path, different predictors may use different alphabets, and so different alphabets will be provided implicitly with the provision of different scoring functions.

4.2.1 Mass Spectra Dereplication

The input to the *de novo* mass spectrometry algorithm is a list of mass spectrometry readings, each with a mass value and an intensity value, where mass gaps represent a potential compound. We then want to convert these values into *sequence tags*, end-to-end lists of possible components. (That is, in a sequence tag, each component must have its gap begin where the previous gap ended.)

Therefore, first, we preprocess our peaks to remove all low-intensity readings, according to some user-chosen threshold (perhaps 5% of the maximum value in a spectrum, for example), and sort them so that they are increasing order of mass. We then want to find gaps in the masses that correspond to the masses in some table of compounds (with some tolerance, either percentile or absolute) and find all end-to-end sequence tags from this.

We can formally model this problem as a **DAG** (Directed Acyclic Graph), a digraph with a topological ordering - that is, each vertex is enumerated and vertices can only have edges to

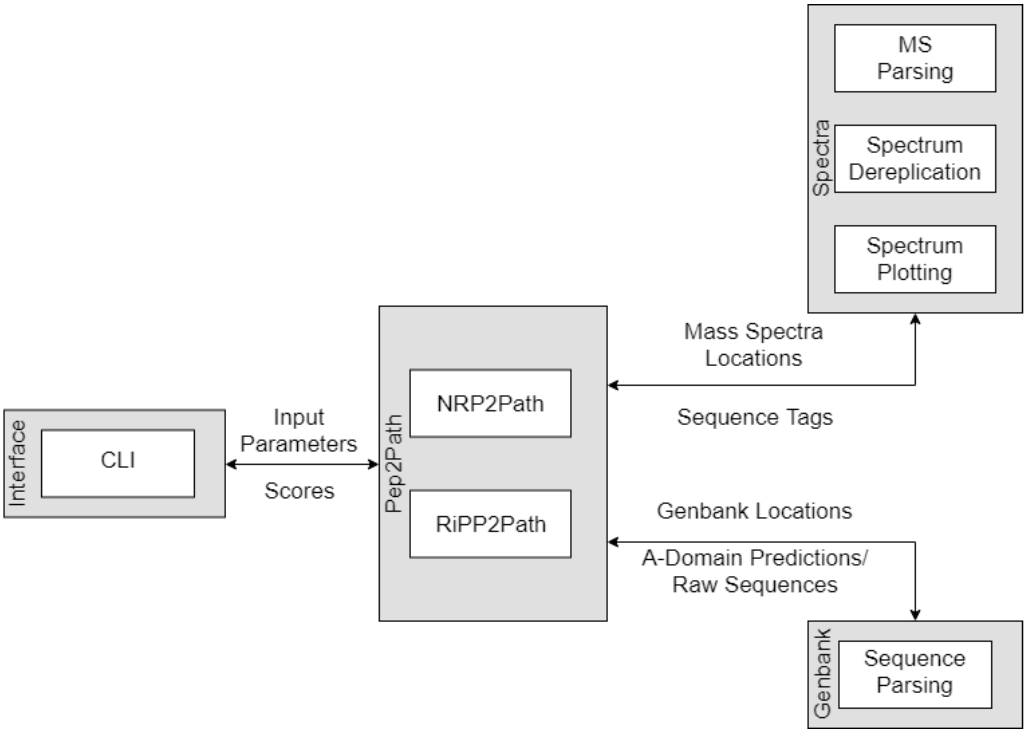

Figure 4.1: *An illustration of our proposed system layout. We write input on the top of arrows, and output on the bottom.*

vertices with higher numbers. (Boecker 2019) (There may be multiple topological orderings - for example consider the case where there are multiple *sources* - nodes with no ingoing edges - in the graph. Either could be labelled as the first node! We specifically use an ordering by increasing mass and direct edges towards higher masses.) Furthermore, edges have a label, which is a list of all potential compounds that could be between the peaks the vertices represent, and these edges and their labels are unknown, but discoverable. In this model, vertices are individual mass spectrometry readings (mass and intensity), and edges are potential compounds from gaps between mass peaks. The problem of finding a sequence tag then corresponds to finding a path through this graph.

We partition this algorithm into two stages: **edge discovery** and **path discovery**. Edge discovery is the process of searching the raw numerical mass values of each peak for gaps corresponding to components in the mass table, in order to build the graph's edge set. Path discovery is then traversing the edges in the edge set in order to form a complete path - and therefore a sequence tag.

To perform edge discovery, we begin by allowing a value to be specified for a mass tolerance. This might be a static value measured in *Da* (Unified Atomic Mass Unit or Dalton), a percentage applied to each mass in the mass table, or a percentage of the maximum mass in the spectra readings. We can then tranform each mass in the mass table into an interval ($mass - mass\_tolerance, mass + mass\_tolerance$). Finally, we iterate through each reading, and subtract it from subsequent readings. If the result lies within the interval, then an edge exists between the peaks with a label corresponding to the compound which the interval belongs to. We repeat this for all values in the mass table in order to find all possible edges.

<note to include edge discovery pseudocode>

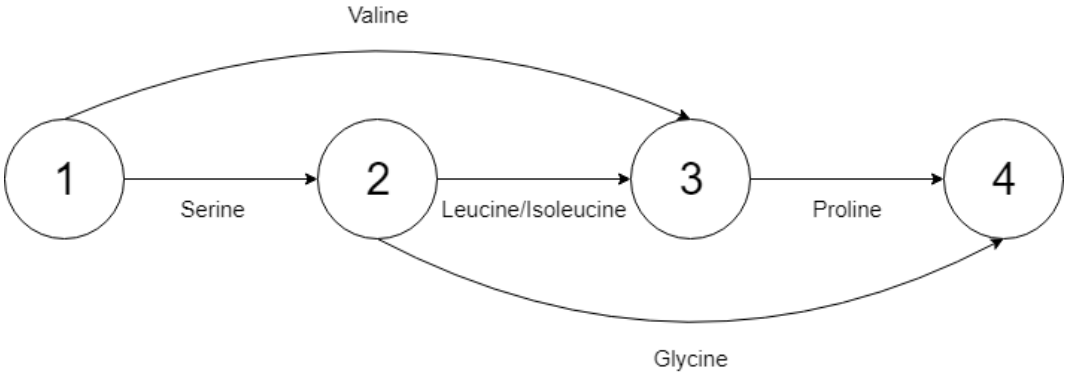

Figure 4.2: *An example illustration of some mass spectra peaks represented as a DAG, with arbitrarily chosen amino acid names for the edges. From this graph there would be the tags (without subtags) Ser-Leu/Ile-Pro, Val-Pro, Ser-Gly.*

To perform path discovery, we perform an exhaustive depth-first search for every peak in the vertex set. (A breadth-first search would also be possible but depth-first is conceptually simpler to understand.) First, we choose a vertex, and denote it our current vertex. So long as our current vertex has edges we haven't yet explored, we choose one of those edges, marking it as explored, and make the other vertex involved in the edge our current vertex, and push our previous vertex onto a stack. We continue this process until we reach a vertex with no unexplored edges. At this point we pop the vertex at the top of our stack, marking it as our current vertex and resume searching for edges. Once we have also exhausted our stack, we choose another vertex we have not already performed this process for, and repeat until we have exhausted all vertices, to get all possible sequence tags.

There are a few obvious optimisations we can make on this process: firstly, we do not need to check for cycles as we normally might in our depth-first search, as by definition our DAG has no cycles. Secondly, the solution space for our tags is very large - $\sum_{len=1}^{n} alphabet\_size^{len}$ specifically, for all tags up to length n, ignoring for now how these tags might be arranged along our peaks. Rather than storing each tag as we find it, we can instead store the longest tag containing this tag as a substring (we refer to such substrings as *subtags* here).

This is done for two reasons: under normal circumstances, we assume if we have found an edge, then unless the data is exceptionally noisy the edge itself is likely meaningful, and furthermore any subtags can be reconstructed by taking n-grams of the parent tag. For each parent tag, this saves us having to store $\frac{n}{2}(n+1)$ subtags. (Consider that a sliding window of length m has $m - n + 1$ possible alignments across a sequence tag of length n, and that we wish to consider the sum of all $1 \leq m \leq n$ and apply the standard formula for the sum of an arithmetic sequence.) For our algorithm, this means saving a sequence tag *only if* we have exhausted all edges and we haven't just popped something from the stack. Additionally, if we visit a vertex as part of path discovery for another vertex, there is no need to begin a tag search beginning from that vertex - any tags discovered this way would be subtags of the tags we have already discovered.

<note to include np-hardness of original problem in its most general form - I can reference the MS book for this>

<note to include path discovery pseudocode>

However, note that this is a relatively naive solution to a well-established problem. We include it here so as to have a complete, end-to-end software pipeline for NRP2Path, but due to our proposed modular structure, it would be relatively easy to replace it with another, more sophisticated mass-

spectrometry module and still take advantage of Pep2Path. For example, an algorithm improving on ours by: using a scoring mechanism to measure the fit between a predicted sequence tag and the real data during edge discovery, eliminating impossible sequences early or by completely different approaches such as optimising a scoring function over the entire scoring space using a genetic algorithm or using a Hidden Markov Model. (Colinge and Bennett 2007) **<note to mention optimisations, such as those in the mass spectrometry book>**

4.2.2 NRP2Path

<note to include NRP2Path pseudocode>

<note to make it clear that we can check both directions of sequence tag in the NRPS algorithm by checking all module configurations against one sequence tag>

4.2.3 RiPP2Path

This algorithm first takes as input a sequence tag to search for, and raw sequence data to search for it in (probably retrieved from a GenBank file). It then translates the raw sequence data into its six-frame translation by sampling the sequence and its reverse complement at starting indices 0, 1 and 2 and translating the bases into amino acids - the translation is done via standard tables, and we omit the details here. Then for each of these translation frames, we align the sequence tag with the frame for every possible position and score the match, using the number of exact matches between the two normalised by the tag length $\frac{no\_matches}{len}$ as a simple scoring mechanism. We then sort our matches, best scores first, and report the strand and position (in nucleotides) for the nth best matches (we allow n to be supplied as a parameter).

<note to include ripp2path pseudocode>

This algorithm bears some resemblance to brute-force string search, but whilst brute force string search searches for an exact match, this algorithm attempts to score every possible substring. We do this so as to provide as much information on the data as possible. For example, suppose that our search tag was one amino acid off, and we could not find it in the data. Then when attempting to find an exact match, we would only have a 'not found' result. But given this extra granularity, a near-total match would show up as the highest score found.

<note: should i move (parts of) this somewhere? the comparison to our method is useful...>

However, as a result, this algorithm can't improve on the $O(n^2)$ complexity of the most naive forms of brute-force string search (i.e. ones without early termination on the first mistmatch when aligning a substring with the query string), because improvements work by skipping some substrings. However, should we wish to search for exact matches, this is a solved problem with standard solutions, such as the linear-time Knuth-Morris-Pratt algorithm (Knuth et al. 1977), the emprically performant Boyer-Moore algorithm (Boyer and Moore 1977) or algorithms such as hybrid KMP/BM algorithms. (Franek et al. 2007) (Baeza-Yates 1989) String search algorithms such as these are also of more use in other areas of bioinformatics more generally as seen in the use of a hybrid KMP/Boyer-Moore in genome profiling in the recent PATSIM (P.Manikandan and D.Ramyachitra 2018) or an alternative exact string-searching algorithm TVSBS, (Thathoo et al. 2006) designed specifically for biological sequence data. However, for most cases of exact string-matching GNU/Linux provides a standard, optimised implementation of Boyer-Moore as part of the *grep* utility.

Alternatively, since we are only searching for high scores, it would be possible to implement an algorithm that searches only for *good* matches and skips bad matches by some means - suggesting how one might do this is beyond the scope of this paper, however.

5 Implementation

5.1 Language Choice

We chose to use Python for our implementation, as it is the *lingua franca* of scientific computing. It offers highly expressive language constructs and useful libraries, such as itertools (Python 3 2008) offering many convenient ways to iterate through combinations, permutations and more, NumPy (NumPy 2006) for high-speed numerical operations and BioPython (BioPython 2000) for bioinformatics. All of these were used in the course of implementing our design. Additionally, a great volume of scientific software is already implemented in Python (indeed, the original Pep2Path is implemented using Python) and many scientists are familiar with Python already, making it easier to interface with the already extant body of work. Particularly, the client's codebase was implemented in Python, so this choice made for easier integration without the need for a language-spanning gateway. Other options like Julia (JuliaLang 2012) exist, but given their lack of maturity when compared to Python we elected not to explore these options.

note to talk about itertools simplifying much of combinatorial maths involved - particularly much of the work of nrp2path can be reduced to two lines (three with optimisation)

note to talk about use of numpy in mass spectrometry - standard methods for a lot of processing, and vectorisation in edge discovery specifically (this is not the performance bottleneck however) - we use a list of dictionaries for path discovery

note to talk about use of bio for genbank sections and ripp2path - std implementation of biological utilities

6 Evaluation

<note to include unit tests – this proves not necessarily ultimate correctness, but that they adhere to certain invariants we expect from our conceptual understanding – particularly notable is the generation of synthetic data and the extraction of the correct pregenerated results from random noise, for all our algorithms>

<note to include scoring function test results – we wish to demonstrate that the maximum value is for a perfect match, and that scores generally get worse as random mutations are introduced into >

<note to include original pep2path results(?) – this shows that we have a platform capable of implementing various kinds of scoring function, and our implementations themselves are (partially) correct but I may not have time to actually collect these results, so in this case I should just outline the procedure>

note to refer back to requirements – such as implementation within the client's codebase – in order to justify why the project is a success

7 Conclusion

A Appendices

7 Bibliography

R. A.Khan. Natural products chemistry: The emerging trends and prospective goals. *Elsevier Science*, 2018.

R. A. Baeza-Yates. String searching algorithms revisited. In *Proceedings of the Workshop on Algorithms and Data Structures*, WADS '89, pages 75–96, London, UK, UK, 1989. Springer-Verlag. ISBN 3-540-51542-9. URL http://dl.acm.org/citation.cfm?id=645928.672525.

BioPython. Biopython official webpage, 2000. URL https://biopython.org/.

K. Blin, M. H. Medema, M. Kazempour, M. A. Fischbach, R. Breitling, E. Takano, and T. Weber. antiSMASH 2.0–a versatile platform for genome mining of secondary metabolite producers. *Oxford Academic*, 2013.

K. Blin, T. Wolf, M. G. Chevrette, X. Lu, C. J. Schwalen, S. A. Kautsar, H. G. S. Duran, E. L. C. de Los Santos, H. U. Kim, M. Nave, J. S. Dickschat, D. A. Mitchell, E. Shelest, R. Breitling, E. Takano, S. Y. Lee, T. Weber, and M. H. Medema. antiSMASH 4.0-improvements in chemistry prediction and gene cluster boundary identification. *Oxford Academic*, 2017.

S. Boecker. Algorithmic mass spectrometry version 0.6.0, 2019. URL https://bio.informatik.uni-jena.de/script-algoms/.

R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, Oct. 1977. ISSN 0001-0782. doi: 10.1145/359842.359859. URL http://doi.acm.org/10.1145/359842.359859.

R. Breitling, A. Ceniceros, A. Jankevics, and E. Takano. Metabolomics for Secondary Metabolite Research. *MDPI*, 2013.

S. Caboche, M. Pupin, V. Leclère, A. Fontaine, P. Jacques, and G. Kucherov. NORINE: a database of nonribosomal peptides. *Oxford Academic*, 2008.

M. G. Chevrette, F. Aicheler, O. Kohlbacher, C. R. Currie, and M. H. Medema. SANDPUMA: ensemble predictions of nonribosomal peptide chemistry reveal biosynthetic diversity across Actinobacteria. *Oxford Academic*, 2017.

J. Colinge and K. L. Bennett. Introduction to Computational Proteomics. *PLOS Computational Biology*, 2007.

F. Franek, C. G.Jennings, and W.F.Smyth. A simple fast hybrid pattern-matching algorithm. *Elsevier Science*, 2007.

A. Ibrahim, L. Yang, C. Johnston, X. Liu, B. Ma, and N. A. Magarvey. Dereplicating nonribosomal peptides using an informatic search algorithm for natural products (iSNAP) discovery. *Proceedings of the National Academy of Sciences of the United States of America*, 2012.

JuliaLang. Julia language official webpage, 2012. URL https://julialang.org/.

R. D. Kersten, Y.-L. Yang, Y. Xu, P. Cimermancic, S.-J. Nam, W. Fenical, M. A. Fischbach, B. S. Moore, and P. C. Dorrestein. A mass spectrometry–guided genome mining approach for natural product peptidogenomics. *Nature Chemical Biology*, 2011.

D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast Pattern Matching in Strings. *Society for Industrial and Applied Mathematics*, 1977.

M. H. Medema, Y. Paalvast, D. D. Nguyen, A. Melnik, P. C. Dorrestein, E. Takano, and R. Breitling. Pep2Path: Automated Mass Spectrometry-Guided Genome Mining of Peptidic Natural Products. *PLOS Computational Biology: Software*, 2014.

H. Mohimani, R. D. Kersten, W. T. Liu, M. Wang, S. O. Purvine, S. Wu, H. M. Brewer, L. Pasa-Tolic, N. Bandeira, B. S. Moore, P. A. Pevzner, and P. C. Dorrestein. Automated Genome Mining of Ribosomal Peptide Natural Products. *American Chemical Society*, 2014.

H. Mohimani, A. Gurevich, A. Shlemov, A. Mikheenko, A. Korobeynikov, L. Cao, E. Shcherbin, L.-F. Nothias, P. C. Dorrestein, and P. A. Pevzner. Dereplication of microbial metabolites through database search of mass spectra. *Nature Communications*, 2018.

NumPy. Numpy official webpage, 2006. URL `http://www.numpy.org/`.

P.Manikandan and D.Ramyachitra. PATSIM: Prediction and analysis of protein sequences using hybrid Knuth-Morris Pratt (KMP) and Boyer-Moore (BM) algorithm. *Elsevier Science*, 2018.

Python 3. Python 3 official itertools documentation, 2008. URL `https://docs.python.org/3/library/itertools.html`.

M. Röttig, M. H. Medema, K. Blin, T. Weber, C. Rausch, , and O. Kohlbacher. Automated Genome Mining of Ribosomal Peptide Natural Products. *American Chemical Society*, 2011.

R. Smith, A. D. Mathis, D. Ventura, and J. T. Prince. Proteomics, lipidomics, metabolomics: a mass spectrometry tutorial from a computer scientist's point of view. *BioMed Central*, 2014.

T. Stachelhaus, H. D. Mootz, and M. A. Marahiel. The specificity-conferring code of adenylation domains in nonribosomal peptide synthetases. *Elsevier Science*, 1999.

R. Thathoo, A. Virmani, S. S. Lakshmi, N. Balakrishnan, and K. Sekar. TVSBS: A fast exact pattern matching algorithm for biological sequences. *Current Science*, 2006.