A Dynamic Programming Algorithm for Finding Highest-Scoring Forbidden-Pairs Paths with Variable Vertex Scores

Matthew A. Goto*

Eric J. Schwabe†

Abstract

In the de novo peptide sequencing problem, output data from a mass spectrometer is used to determine the peptide whose fragmentation yielded that output. Candidate peptides are determined by finding forbidden-pairs paths in a spectrum graph constructed from the mass spectrometer data, assigning scores to vertices and/or edges in order to favor higher-scoring peptides. Chen et al. gave an algorithm to find the highest-scoring forbidden-pairs path in such a graph. However, in some scoring models, vertex scores may vary depending on which incident edges are used in the path, ruling out the use of this algorithm. We give an algorithm to solve the highest-scoring forbidden-pairs paths problem when vertex scores can vary in this way that runs in $O(n^2d^3)$ time on a graph with n forbidden pairs and a maximum vertex degree of d. We are currently working on a Java implementation of this algorithm that we plan on incorporating into the Illinois Bio-Grid Desktop.

*School of CTI, DePaul University, 243 S Wabash Ave, Chicago IL 60604. mattgoto@gmail.com.

†School of CTI, DePaul University, 243 S Wabash Ave, Chicago IL 60604. eschwabe@cs.depaul.edu.

1 Introduction

The problem of finding the highest-scoring forbidden-pairs path with variable vertex scores arises when we are trying to determine the most likely amino acid sequence to have generated a particular output from a tandem mass spectrometer (MS/MS). There are two typical ways of determining this sequence: Finding the best match from a database of known amino acid sequences, and determining the sequence based solely on the MS/MS data, which is called *de novo sequencing*. The database matching approach has some drawbacks. It requires additional information, such as the genome from which the sample came, in order to be able to reliably determine the sequence, and this additional information is not always available. Furthermore, if we have a novel amino acid sequence which has never been identified before, doing a database search will prove fruitless. For these reasons, de novo sequencing remains important even when databases are available. The problem of finding the highest-scoring forbidden-pairs path with variable vertex scores arises from the de novo sequencing problem when certain sequence scoring methods are used.

An amino acid is a molecule that has the molecular formula $H_2N\text{-}CH(R)\text{-}COOH$, where R is a side chain off of the central carbon atom. There are twenty different side chains, each yielding a different amino acid. Often each amino acid is represented by a one-letter code. A peptide is a sequence of fewer than fifty amino acids and is often written as a list of the letters for its amino acids. A protein is a longer sequence of amino acids. In both peptides and proteins, the order of the amino acids is significant. For example, the peptide GPG is different from GGP.

The input to the *de novo peptide sequencing problem* is a list of peaks representing an output spectrum. Each peak has an intensity (or abundance) and a value representing its mass/charge ratio. This input is generated using a combination of chemical and mechanical techniques. A protein is often digested (broken into smaller parts) by some enzyme into smaller peptides. The digested material is not of a uniform mass, so a technique for separating the digested material by mass is used. A liquid chromatograph (LC) is one such method. A single drip (containing molecules with a single, but as yet unknown, mass) from the LC is input into a mass spectrometer (MS), which determines the mass/charge and intensity of the input. If there is a sufficient amount of the material in the drip, the material in the drip is fragmented further by one of a variety of processes, one of which is called collisionally induced dissociation. The charge and composition of the peptide determines where the breakages occur. The peptide fragments are then placed in a second mass spectrometer for further analysis. The output of the second mass spectrometer, called a tandem mass spectrometer (MS/MS) spectrum, is used as the input to the de novo peptide sequencing problem and is comprised of pairs of mass/charge and intensity values for the peptide fragments. These pairs are called peaks as stated earlier. In addition to the list of peaks, the mass of the peptide (as determined by the initial MS information) and the charge of the peptide are given as inputs to the problem.

There are various types of fragmentation that can occur as a result of the collisionally induced dissociation. When the fragmentation occurs between two amino acids in the original peptide, a prefix and a suffix of the peptide are created. The prefix is called a b-ion and the suffix is called a y-ion. These are the most common types of fragmentation, but there are additional types. A b-ion minus water (b-H_2O) or a y-ion minus water (y-H_2O) is formed when a b-ion or y-ion loses a water molecule. Similarly, a b-ion or y-ion minus ammonia (b-NH_3 or y-NH_3) results when a b or y-ion loses an ammonia molecule. Sometimes fragmentation does not occur exactly between two amino acids. An a-ion or an x-ion are examples of such fragmentation. An a-ion is a b-ion that is missing a carbon and oxygen atom from its end. A x-ion is a y-ion that has an extra carbon and oxygen atom on its front.

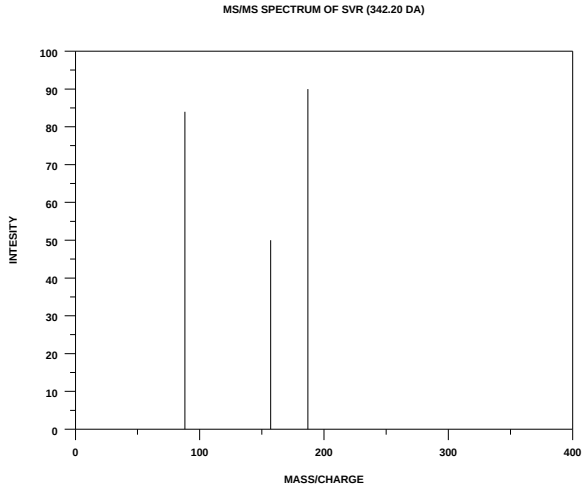

Figure 1: A hypothetical MS/MS spectrum for the peptide SVR.

The goal of the de novo sequencing problem is to find the original peptide from just the MS/MS output. The solution to the problem is complicated by the fact that MS/MS output is often noisy, i.e. it has peaks that are not from an actual fragment of the peptide but rather have some other cause, such as a contaminant or machine error. It is complicated further by the fact that there may have been no instances of fragmentation between some pair of adjacent amino acids.

Section 2 will discuss previous work in the field of the de novo sequencing problem and how the problem of the highest-scoring forbidden-pairs path with variable vertex scores arose. Section 3 will discuss our algorithm to solve the problem of finding highest-scoring forbidden-pairs paths with variable vertex scores. Section 4 will discuss our work on implementing our algorithm and future areas for research and investigation.

2 Previous Work

The typical approach used to solve the de novo peptide sequencing problem is to construct a spectrum graph from a subset of the peaks, and find some path in the graph. Usually the desired path is the highest scoring path among some subset of the possible paths. Scores for edges and/or vertices are determined by some scoring method and the score of a path is the sum of the scores of its edges and vertices.

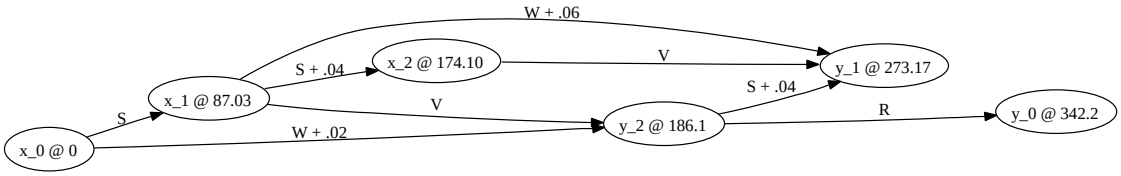

Figure 2: The spectrum graph from Figure 1 as created by Chen et al.

2.1 Spectrum Graph Construction

A spectrum graph (see, e.g., Dancik et al. [2]) is constructed by creating a vertex for each peak in the input data and for each type in a set of fragmentation ion types (such as b-ion, y-ion, a-ion, b-H_2O-ion, etc.) The i^{th} ion type would modify a peak's mass by a set amount, δ_i. So for each of the n peaks in the input data with mass p_i, a vertex is created for each fragmentation ion type with a mass of $p_i + \delta_i$. There are two additional vertices in the spectrum graph: the vertex associated with a mass of 0 and the vertex associated with the mass of the peptide. Thus, if the ion type set had five values and there were fifty peaks in the input data, there would be $2 + 5 \cdot 50 = 252$ vertices in the spectrum graph. There is one caveat with the creation of vertices. If two or more vertices are created with the same mass, then they are merged into a single vertex.

A directed edge connects two vertices in the spectrum graph if the difference in the vertices' masses is the mass of an amino acid. (Some constructions also include edges between vertices whose masses differ by the sum of two or more amino acids.) The tail of the edge is the vertex with the smaller mass and the head is the vertex with the larger mass. The resulting spectrum graph is a directed acyclic graph. (For example, see Figures 1 and 2, where the construction uses the two fragmentation ion types of b-ion and y-ion so that the ith peak in the spectrum generates two vertices x_i and y_i, and two pairs of vertices have been merged to obtain the final set of six vertices.)

2.2 Forbidden-Pairs Paths

It should be noted that since each peak in the spectrum graph can produce multiple vertices, at most one of these vertices can be included in any path through the spectrum graph that is intended to represent a peptide, since the vertices represent mutually exclusive interpretations of the peak from which they were created. These sets of vertices are called *forbidden sets* of vertices or in the case of there only being two vertices in each forbidden set, *forbidden pairs*. A path that uses at most one vertex from each forbidden pair is called a *forbidden-pairs path*; each such path represents a peptide that may have generated the input data. (This is an example of what Dancik et al. [2] called an *antisymmetric path*.)

Chen et al. [1] gave a dynamic programming algorithm to solve the highest-scoring forbidden-pairs path problem for a spectrum graph. When constructing the spectrum graph, they only create vertices for a b-ion and a y-ion interpretation of each peak. Thus they have forbidden pairs of vertices in their spectrum graph. Their algorithm is as follows, where E is the adjacency matrix of the spectrum graph and s is the scoring function for its edges [1]:

[1] In line 5 of the algorithm, they start at $i = 1$. We believe this to be a typographical error, as there are counterexamples where the algorithm would produce incorrect results if i started at 1 instead of 0.

Figure 3: Line 3 from Chen et al.'s algorithm - starting a new pair of paths x_0 and y_j to y_0.

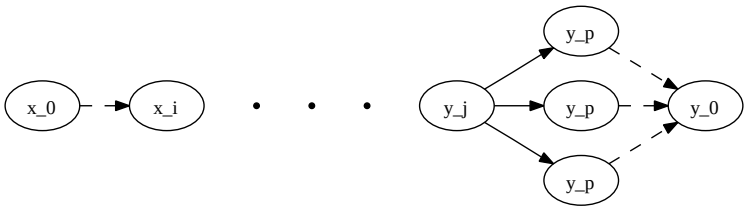

Figure 4: Line 5a from Chen et al.'s algorithm - extending an existing pair of paths from $x_0 \ldots x_i$ and $y_p \ldots y_0$ to $x_0 \ldots x_i$ and $y_j \ldots y_0$.

Algorithm Compute-$Q(G)$

1. Initialize $Q(i,j) = 0$ for all $0 \leq i,j \leq n$;
2. For $j = 1$ to n
3. If $E(y_j, y_0) = 1$, then $Q(0,j) = \max\{Q(0,j), s(y_j, y_0)\}$;
4. If $E(x_0, x_j) = 1$, then $Q(j,0) = \max\{Q(j,0), s(x_0, x_j)\}$;
5. For $i = 0$ to $j-1$
   (a) For every $E(y_j, y_p) = 1$ and $Q(i,p) > 0$, $Q(i,j) = \max\{Q(i,j), Q(i,p) + s(y_j, y_p)\}$;
   (b) For every $E(x_p, x_j) = 1$ and $Q(p,i) > 0$, $Q(j,i) = \max\{Q(j,i), Q(p,i) + s(x_p, x_j)\}$;

The purposes of lines 1-5 are fairly clear: They initialize the elements of the two-dimensional table Q to contain either 0, the score of an edge from y_j to y_0 (line 3), or the score of an edge from x_0 to x_j (line 4). (See Figure 3.) Lines 5a and 5b require more explanation. They extend an existing highest scoring pair of paths (either x_0 to x_i and y_p to y_0 in line 5a or x_0 to x_p and y_i to y_0 in line 5b) to include another vertex, (either y_j for line 5a or x_j for line 5b). If an extension can be performed, the extended pairs of paths are x_0 to x_i and y_j to y_0 (for line 5a) or x_0 to x_j and y_i to y_0 (for line 5b). Note that each line extends only one of the pair of paths, either the x path or the y path, but not both. In other words, line 5a extends the y path from $y_p \ldots y_0$ to $y_j \ldots y_0$, and line 5b extends the x path from $x_0 \ldots x_p$ to $x_0 \ldots x_j$. (See Figure 4.) Finally, the greatest sum of the score of the additional edges plus the previous value in Q for that path is stored in $Q(i,j)$ or $Q(j,i)$. In short, for all $0 \leq i,j \leq n$, $Q(i,j)$ is constructed to hold the sum of the score of the highest scoring pair of paths $x_0 \ldots x_i$ and $y_j \ldots y_0$ where at most one vertex from each forbidden pair is used in the pair of paths. $Q(i,j) = 0$ indicates that there is no such pair of paths.

Chen et al.'s algorithm runs in $O(VE)$ time, where $V = 2n + 2$ is the number of vertices in the graph when n is the number of forbidden pairs in the graph, and E is the number of edges in the graph.

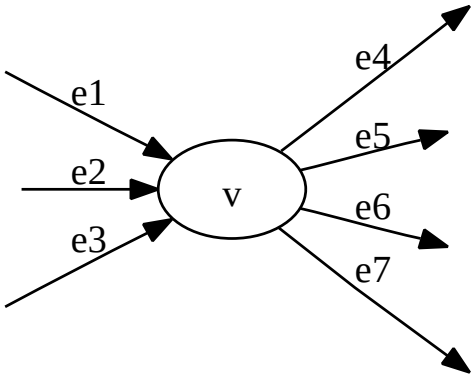

Figure 5: A vertex with a variable vertex score – i.e., the vertex's score depends upon the edges taken entering and exiting it.

2.3 Path Scoring

Scores for vertices and/or edges are chosen to increase the likelihood of finding the correct peptide by influencing which path is the most desirable, i.e., the highest-scoring path. Various scoring models have been proposed. Chen et al. assumed that the scoring function s was supplied as input, allowing them to focus on the algorithm for finding the highest-scoring forbidden-pairs path given that scoring function. Dancik et al. [2] gave a scoring function that assigned "a premium for present ions [vertices] and a penalty for missing ions [vertices]."

In Pevzner and Frank's paper on the PepNovo system [3], they proposed a new scoring model based on the probabilities of different types of peptide fragmentation that yields vertex scores that depend on which of the vertices' incident edges are used in the path. For this *variable vertex-scoring* method, the algorithm of Chen et al. does not apply. PepNovo claims to use a variant of the Chen et al. algorithm, but the details are not discussed in the paper, nor are they well documented in their source code.

Here, we present a dynamic programming algorithm for a spectrum graph with n forbidden pairs and maximum vertex degree d that will find the highest-scoring forbidden-pairs path in $O(n^2d^3)$ time. This is the first algorithm that has been proved to find the highest-scoring forbidden-pairs path when the vertices' scores depend upon the incident edges chosen for each vertex in the path.

3 Our Result

3.1 Spectrum Graph Construction

In order to accommodate variable vertex scores, each vertex of the spectrum graph is replaced with a *super-vertex* containing a complete bipartite graph.

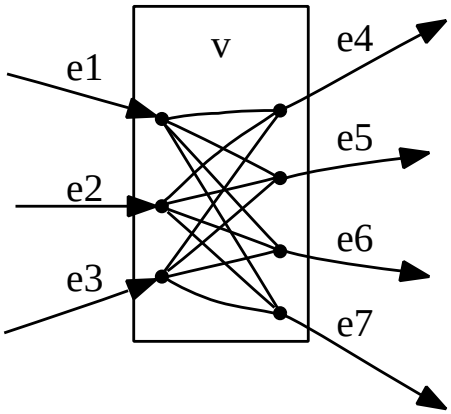

Figure 6: The transformation of the vertex in Figure 5.

Each bipartite graph consists of a *left set* of vertices with a vertex for each incoming edge of the original vertex, and a *right set* of vertices with a vertex for each outgoing edge of the original vertex. In the super-vertex for v, we call these sets $L(v)$ and $R(v)$. $L(x_0)$ and $R(y_0)$ are each defined to consist of a single vertex, named s and t, respectively. All edges in the complete bipartite graph are directed from left to right.

For each original vertex a, let e be the number of incoming edges. Exactly one of these incoming edges will be incident to each left vertex in the bipartite graph within a's super-vertex. Then, let f be the number of outgoing edges. Exactly one of these outgoing edges will be incident to each right vertex in the bipartite graph within a's super-vertex. So, $|L(a)| = e$ and $|R(a)| = f$. (See Figures 5, 6, and 7.)

To find the score from u' in $L(b)$ to v' in $R(b)$ within a super-vertex b: Let a be the super-vertex such that u' is the lone successor of some vertex in $R(a)$. Let c be the super-vertex such that v' is the lone predecessor of some vertex in $L(c)$. Then the score of (u', v'), called $w(u', v')$, is defined to be the score of the original vertex b when the incident edges are (a, b) and (b, c).

For each edge (u, v) in the original graph with score x, the edge from a vertex $u' \in R(u)$ to a vertex $v' \in L(v)$ will be given score $w(u', v') = x$ also.

This technique of replacing vertices in the original graph allows us to take a graph where the vertex scores depend on which incident edges are chosen in the path and replace it with a graph with fixed edge scores and no vertex scores. If we can find a highest-scoring forbidden-pairs path in the transformed graph (where the forbidden pairs are now pairs of super-vertices), we will be able to construct a highest-scoring forbidden-pairs path in the original graph by just taking the sequence of super-vertices visited.

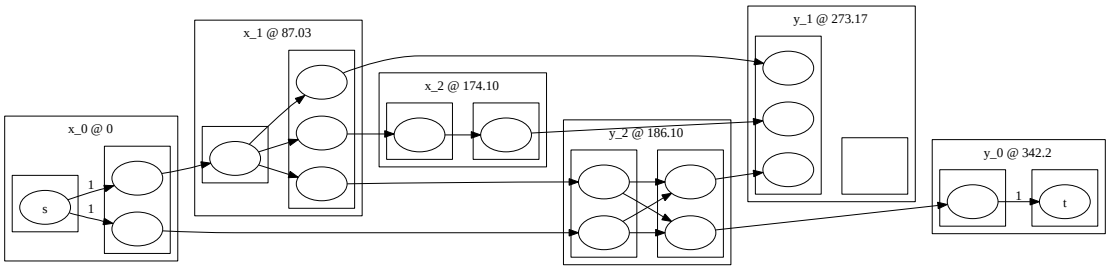

Figure 7: The result of transforming the spectrum graph in Figure 2.

3.2 Our Algorithm

Previous algorithms for finding forbidden-pairs paths cannot be used to solve the problem when variable vertex scores are added, as each fails to address some aspect of the problem. The algorithm proposed by Chen et al. does not handle the variable vertex scores; it assumes that vertex scores are constant and do not depend upon incident edges. The algorithm used by Dancik et al. does not apply here since their algorithm determines the highest scoring forbidden-pairs path given sets of mutually exclusive vertices. To find the highest scoring forbidden-pairs path on the transformed graph, which has constant edge scores instead of variable vertex scores, the algorithm would need to choose one vertex from each of the two sets in a super-vertex. This pair of vertices are mutually exclusive from a second pair of vertices from the sets in the forbidden pair super-vertex. That is, the Dancik et al. algorithm would need to pick a vertex from each of the L and R sets from either the x or the y forbidden pair sets but not both. It does not do this, and so cannot be used on the transformed graph. (The algorithm of Chen et al. has the same problem.) Further, since we have not been able to find the algorithm used by Dancik et al. for determining the highest scoring forbidden-pairs path, we cannot say whether their algorithm would be able to properly handle variable vertex scores on the untransformed graph.

We find the highest-scoring forbidden-pairs path in the transformed spectrum graph using dynamic programming, by extending the approach used by Chen et al., as follows. We fill in a two-dimensional table Q consisting of objects $Q(i,j), 0 \leq i,j \leq n$ where n is the number of forbidden pairs. Each $Q(i,j)$ contains:

- maxWeight: the maximum score of all possible pairs of paths from s in $L(x_0)$ to some u in $L(x_i)$ and from some v in $R(y_j)$ to t in $R(y_0)$ that contain at most one super-vertex from each forbidden pair.
- weights: a map with key set a subset of $L(x_i) \times R(y_j)$, where weights(u,v) is an object containing three things:
  - score: the score of the max-score paths from s to u and from v to t that contain at most one super-vertex from each forbidden pair.
  - backtrackIndices: a pair of indices in Q which will be one of $(0,0), (i,p)$, or (p,j). This is used for backtracking.
  - backtrackKey: a key into the weights map from $Q(0,0)$, $Q(i,p)$, or $Q(p,j)$ depending upon if the pair of indices is $(0,0)$, (i,p), or (p,j). This is also used for backtracking.

If no paths from s to u and from v to t that contain at most one super-vertex from each forbidden pair have been discovered, or none exist, weights(u, v) is not assigned a value in the map.

The table is constructed as follows:

1. For all i and j between 0 and n
   $Q(i, j)$.maxWeight $= -\infty$ (except for $Q(0, 0)$.maxWeight $= 0$)
   $Q(i, j)$.weights $= \{\}$ (except for $Q(0, 0)$.weights $= \{((s, t), (0, (0, 0), (s, t)))\}$)

2. For $j = 1$ to n

3. If an edge exists between some vertex u in $R(y_j)$ and v in $L(y_0)$ and $w(u, v) + w(v, t) > Q(0, j)$.maxWeight, set
   $Q(0, j)$.maxWeight $= w(u, v) + w(v, t)$
   $Q(0, j)$.weights$(s, u) = (w(u, v) + w(v, t), (0, 0), (s, t))$
   (See Figure 8.)

4. If an edge exists between some vertex u in $R(x_0)$ and v in $L(x_j)$ and $w(s, u) + w(u, v) > Q(j, 0)$.maxWeight, set
   $Q(j, 0)$.maxWeight $= w(s, u) + w(u, v)$
   $Q(j, 0)$.weights$(v, t) = (w(s, u) + w(u, v), (0, 0), (s, t))$

5. For $i = 0$ to $j - 1$

   a. For each u in $L(x_i)$ and v in $R(y_j)$, compute $Q(i, j)$.weights(u, v) as follows:

      For every super-vertex y_p such that there is an edge from v to some a in $L(y_p)$ and such that $Q(i, p)$.maxWeight $> -\infty$, compute the max over all b in $R(y_p)$ such that $Q(i, p)$.weights(u, b) exists of $Q(i, p)$.weights$(u, b) + w(v, a) + w(a, b)$. If there are no such y_p or b, (u, v) is not given a value in the map at this time. Otherwise, call the resulting maximum uvWeight, and let (u, b^*) be the key to $Q(i, p)$.weights that produces this maximum value. $Q(i, j)$.weights$(u, v) =$ (uvWeight,$(i, p), (u, b^*)$). (See Figure 9.)

   If any keys in $Q(i, j)$.weights have a value, then do the following: Let (u^*, v^*) be the key of the largest value of score in the map $Q(i, j)$.weights. Set $Q(i, j)$.maxWeight $=$ $Q(i, j)$.weights(u^*, v^*).score.

   b. For each u in $L(x_j)$ and v in $R(y_i)$, compute $Q(j, i)$.weights(u, v) as follows:

      For every super-vertex x_p such that there is an edge from some a in $R(x_p)$ to u and such that $Q(p, i)$.maxWeight $> -\infty$, compute the max over all b in $L(x_p)$ such that $Q(p, i)$.weights(b, v) exists of $Q(p, i)$.weights$(b, v) + w(b, a) + w(a, u)$. If there are no such x_p or b, (u, v) is not given a value in the map at this time. Otherwise, call the resulting maximum uvWeight, and let (b^*, v) be the key to $Q(p, i)$.weights that produces this maximum value. $Q(j, i)$.weights$(u, v) =$ (uvWeight,$(p, i), (b^*, v)$).

   If any keys in $Q(j, i)$.weights have a value, then do the following: Let (u^*, v^*) be the key of the largest value of score in the map $Q(j, i)$.weights. Set $Q(j, i)$.maxWeight $=$ $Q(j, i)$.weights(u^*, v^*).score.

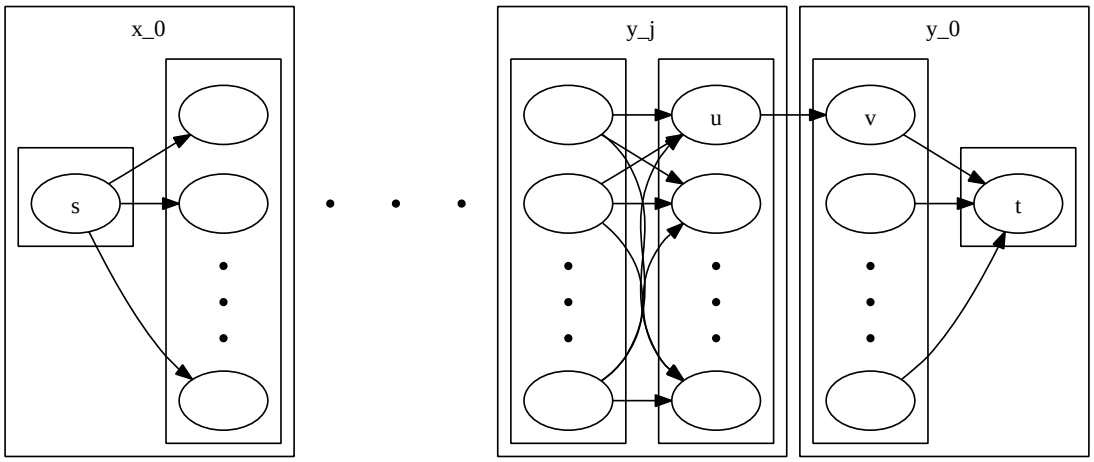

Figure 8: Line 3 from our algorithm - starting a new pair of paths x_0 and y_j to y_0. Similar to Figure 3.

To get the score of the highest-scoring path from Q, called $W(i, j)$, use the following algorithm:

6. For $i = 0$ to n

7. For $j = 0$ to n

8. If there is an edge (a, b) from vertex a in $R(x_i)$ to vertex b in $L(y_j)$ and the map $Q(i, j)$.weights has more than zero keys, compute

   $W(i, j)$.pathWeight = the maximum of
   $Q(i, j)$.weights(u, v).score + $w(u, a) + w(a, b) + w(b, v)$ over all keys (u, v) in $L(x_i) \times R(y_j)$ for which $Q(i, j)$.weights(u, v) exists.

   $W(i, j)$.backtrack = (u^*, v^*) where (u^*, v^*) is the key to $Q(i, j)$.weights that maximized the value of $W(i, j)$.pathWeight.

   Otherwise (if there is no (a, b) edge or there are zero keys in $Q(i, j)$.weights), $W(i, j)$.pathWeight and $W(i, j)$.backtrack are given no value.

   (See Figure 10.)

9. The largest value of $W(i, j)$.pathWeight, found at location (i^*, j^*), is the score of the highest-scoring forbidden-pairs path from s to t. If no $W(i, j)$.pathWeight has a value, there are no forbidden-pairs paths.

Backtracking to compute the highest-scoring forbidden-pairs path is relatively straightforward going from (i^*, j^*) to $(0, 0)$ using $W(i^*, j^*)$.backtrack and the backtrackIndices and backtrackKey fields of the $Q(i, j)$.weights references, as follows:

If the best crossover is at (i^*, j^*), let a be the vertex in $R(x_{i^*})$ that connects x_{i^*} to y_{j^*}, and b be the vertex in $L(y_{j^*})$ that connects x_{i^*} to y_{j^*}. Let (u, v) be $W(i^*, j^*)$.backtrack. Begin with a

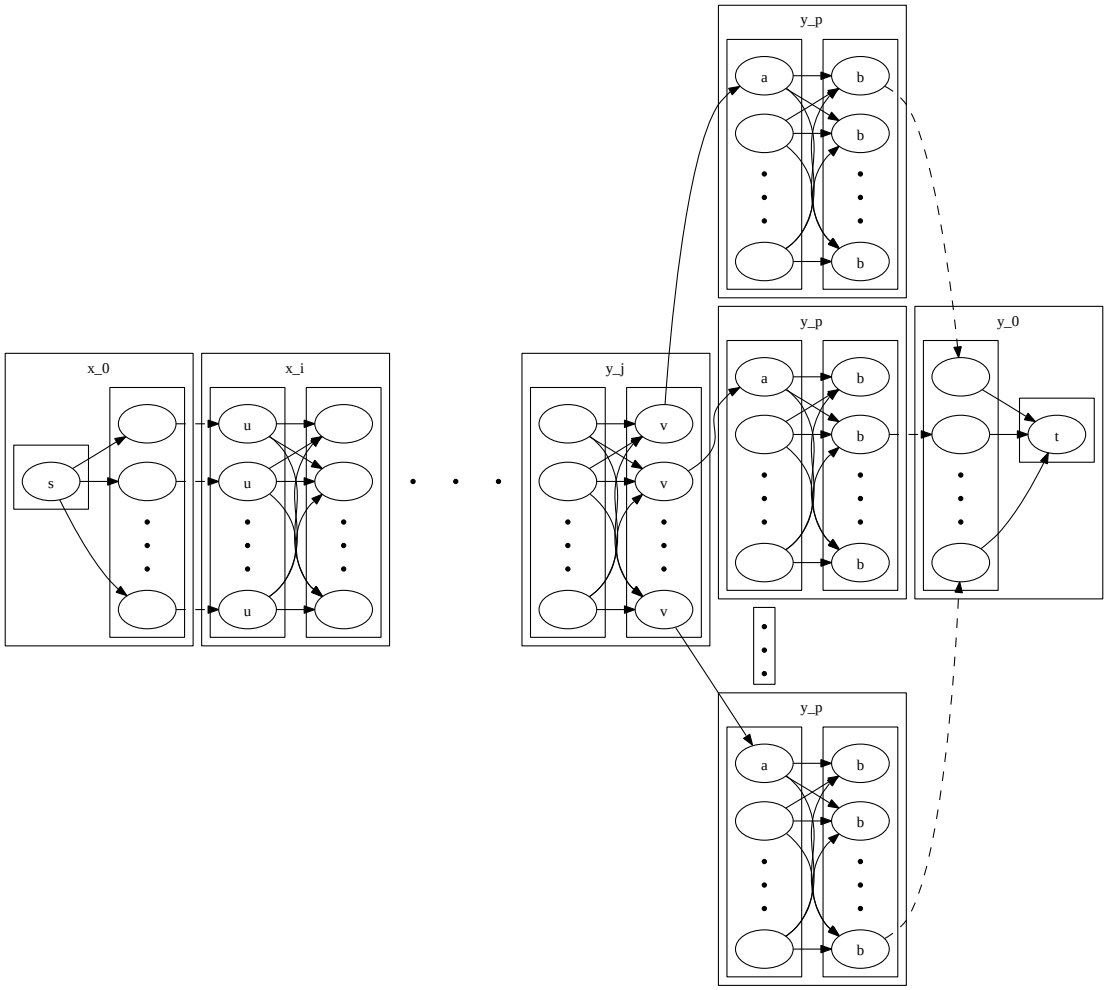

Figure 9: Line 5a from our algorithm - extending an existing pair of paths from $x_0 \dots x_i$ and $y_p \dots y_0$ to $x_0 \dots x_i$ and $y_j \dots y_0$. Similar to Figure 4.

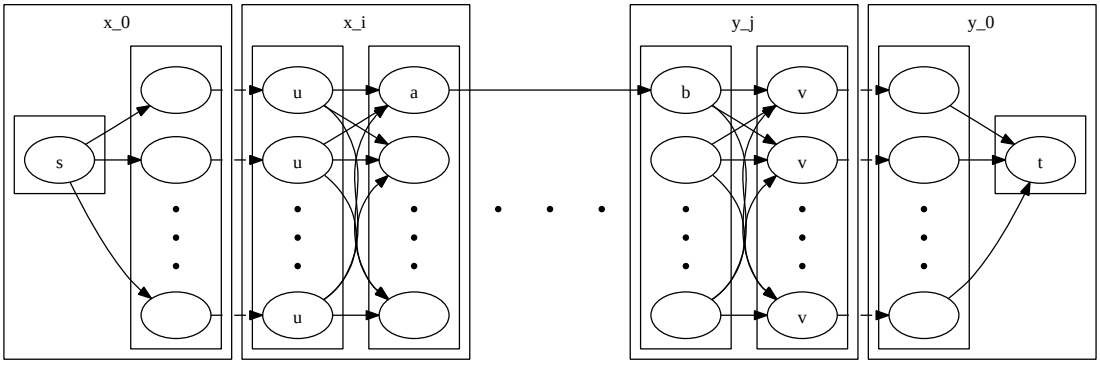

Figure 10: Line 8 from our algorithm - crossing over from x_i to y_j.

path P consisting of just the edges ua, ab, and bv. Let the current indices (i, j) be (i^*, j^*) and the current key be (u, v), initially. Then do the following:

From the current indices (i, j) and key (u, v), find the values of $Q(i, j)$.weights(u, v).backtrackIndices and $Q(i, j)$.weights(u, v).backtrackKey – by construction, they will be one of $(0, 0)$ and $(s, t), (i, p)$ and (u, b) for some $p < j$ and where (u, b) is an element of $L(x_i) \times R(y_p)$, or (p, j) and (b, v) for some $p < i$ and where (b, v) is an element of $L(x_p) \times R(y_j)$. In the first case, i, j, or both must be equal to zero. If both are equal to zero, we are done and P is the desired highest scoring forbidden-pairs path. If $i = 0$, let $p = 0$ and $b = t$ and follow the steps for the second case. If $j = 0$, let $b = s$ and $p = 0$ and follow the steps for the third case.

In the second case, by the construction of the forbidden-pairs path, v is the current last vertex in P. Let a be the vertex in $L(y_p)$ which connects v and b. Add to the back of P the edges va followed by ab. Replace the current indices (i, j) with (i, p). Replace the current key (u, v) with (u, b).

In the third case, by the construction of the forbidden-pairs path, u is the current first vertex in P. Let a be the vertex in $R(x_p)$ which connects b and u. Add to the front of P the edges au followed by ba. Replace the current indices (i, j) with (p, j). Replace the current key (u, v) with (b, v).

Repeat this process until the current pair is $(0, 0)$.

To establish the running time of our algorithm, we note that a spectrum graph with n forbidden pairs and maximum degree d has $V = 2n + 2$ vertices and $E \leq dV$ edges, and the resulting transformed spectrum graph has $V' < 2dn = O(dn)$ vertices and $E' \leq d^2V + E = O(d^2V)$ edges.

Step 1 takes $\Theta(n^2)$ steps, as there are n^2 elements in the table Q. Steps 3 and 4 are each executed n times, and each execution takes $O(d)$ steps, as we may have to check as many as d outgoing edges from y_j (or incoming edges to x_j). This assumes that the transformed graph is represented as an adjacency matrix, and that the get and put operations for the weights map take constant time. Steps 5a and 5b are each executed $\Theta(n^2)$ times, and each execution takes $O(d^3)$ steps. For both Step 5a and Step 5b, there are at most d values of u that must be considered, at most d values of v that must be considered, and for each particular u and v, up to d values of b over which we must take the maximum path score. Finally, Steps 6-9 take a total of $O(n^2d^2)$ steps, as there are $\Theta(n^2)$ iterations and Step 8 takes $O(d^2)$ steps to consider the possible values of a and b. (The value needed in Step 9 can be computed during the loop with an extra constant time per iteration.)

The dominant term comes from Steps 5a and 5b, which yield an overall running time of $O(n^2d^3)$.

3.3 Proof of Correctness

First, we establish the correctness of the dynamic programming construction, as follows:

THEOREM: For any i and j between 0 and $n, i < j$, and for $(i, j) = (0, 0)$, the following eight-part predicate $M(i, j)$ holds:

1. When the object $Q(i, j)$ is computed by the algorithm above, it is the case that

   (a) for every u in $L(x_i)$ and v in $R(y_j)$, weights(u, v).score is the value of the maximum possible score of paths from s to u and from v to t that contain at most one super-vertex from each forbidden pair (if any such paths exist – otherwise it has no value), and

(b) for every u in $L(x_i)$ and v in $R(y_j)$, weights(u, v).backtrackIndices is the pair of indices in Q which is one of $(0, 0), (i, p)$, or (p, j) and results in the maximum possible value for weights(u, v).score (if weights(u, v).score is given a value – otherwise it has no value), and

(c) for every u in $L(x_i)$ and v in $R(y_j)$, weights(u, v).backtrackKey is the key into the weights map of Q(weights(u, v).backtrackIndices) which results in the maximum possible value for $Q(i, j)$.weights(u, v).score (if weights(u, v).score is given a value – otherwise it has no value), and

(d) maxWeight is the value of the maximum possible score of paths from s to any vertex in $L(x_i)$ and from any vertex in $R(y_j)$ to t that contain at most one super-vertex from each forbidden pair (if any such paths exist – otherwise it will be 0).

2. When the object $Q(j, i)$ is computed by the algorithm above, it is the case that

(a) for every u in $L(x_j)$ and v in $R(y_i)$, weights(u, v).score is the value of the maximum possible score of paths from s to u and from v to t that contain at most one super-vertex from each forbidden pair (if any such paths exist – otherwise it has no value), and

(b) for every u in $L(x_j)$ and v in $R(y_i)$, weights(u, v).backtrackIndices is the pair of indices in Q which is one of $(0, 0), (i, p)$, or (p, j) and results in the maximum possible value for weights(u, v).score (if weights(u, v).score is given a value – otherwise it has no value), and

(c) for every u in $L(x_j)$ and v in $R(y_i)$, weights(u, v).backtrackKey is the key into the weights map of Q(weights(u, v).backtrackIndices) which results in the maximum possible value for $Q(i, j)$.weights(u, v).score (if weights(u, v).score is given a value – otherwise it has no value), and

(d) maxWeight is the value of the maximum possible score of paths from s to any vertex in $L(x_j)$ and from any vertex in $R(y_i)$ to t that contain at most one super-vertex from each forbidden pair (if any such paths exist – otherwise it will be 0).

PROOF: We proceed by induction on (i, j).

Base case: We note that by the initialization of $Q(0, 0)$ defined in the algorithm, the conditions a), b), c), and d) above are already satisfied for $i = 0$ and $j = 0$.

Inductive step: Let i and j between 0 and $n, i < j$, be given. Assume that $M(i, j)$ holds for all (i', j') where either $j' < j$ or ($j' = j$ and $i' < i$). (Note that this represents all objects $Q(i', j')$ and $Q(j', i')$ that are computed before $Q(i, j)$ and $Q(j, i)$.)

Proof of 1a: For i and j, let u in $L(x_i)$ and v in $R(y_j)$ be given. Let P be the max-score pair of forbidden-pairs paths from s to u and from v to t. Call the two vertices that immediately follow v in P a and b, and let y_p be the super-vertex for which a is in $L(y_p)$ and b is in $R(y_p)$. Since P is the max-score pair of forbidden-pairs paths, its sub-paths from s to u and from b to t must have score $Q(i, p)$.weights(u, b).score (by the inductive hypothesis). The total score of P is thus $Q(i, p)$.weights(u, b).score + $w(v, a)$ + $w(a, b)$. As Step 5a of the algorithm computes weights(u, v).score as the largest value of this expression over all b in $R(y_p)$ such that $Q(i, p)$.weights(u, b) exists (unless $i = 0$ and $p = 0$, in which case it is computed by Step 3 together with Step 5a), we have that weights(u, v).score must be the score of P, which is the value of the maximum possible score of paths from s to u and from v to t that contain at most one super-vertex from each forbidden pair.

Proof of 1b: For i and j, let u in $L(x_i)$ and v in $R(y_j)$ be given. Let P be the max-score pair of forbidden-pairs paths from s to u and from v to t. It follows from 1a: that weights(u, v).score is the score of P. Call the two vertices that immediately follow v in P a and b, and let y_p be the super-vertex for which a is in $L(y_p)$ and b is in $R(y_p)$. As Step 5a of the algorithm sets weights(u, v).backtrackIndices to (i, p) (or Step 5a combined with Step 3 in the case where $i = 0$ and $p = 0$), we have that weights(u, v).backtrackIndices is in $\{(0, 0), (i, p), (p, j)\}$ and results in the maximum possible value for weights(u, v).score given the restrictions on it.

Proof of 1c: For i and j, let u in $L(x_i)$ and v in $R(y_j)$ be given. Let P be the max-score pair of forbidden-pairs paths from s to u and from v to t. It follows from 1a: that weights(u, v).score is the score of P. Call the two vertices that immediately follow v in P a and b, and let y_p be the super-vertex for which a is in $L(y_p)$ and b is in $R(y_p)$. It follows from 1b: that weights(u, v).backtrackIndices are the indices in Q which give the maximum value for weights(u, v).score for paths s to u and v to t with at most one super-vertex from each forbidden pair. So it follows that one of the keys from the weights map in $Q(Q(i, j)$.weights(u, v).backtrackIndices) must be used to calculate the score for $Q(i, j)$.weights(u, v).score. Further, since Step 5a (or Step 3 together with Step 5a, in the case where $i = 0$ and $p = 0$), assigns $Q(i, j)$.weights(u, v).backtrackKey the key which maximizes the weights(u, v).score, we have the desired result, namely, weights(u, v).backtrackKey is the key into the weights map of Q(weights(u, v).backtrackIndices) which results in the maximum possible value for $Q(i, j)$.weights(u, v).score (if weights(u, v).score is given a value – otherwise it has no value).

Proof of 1d: Since maxWeight is set to be the maximum value in the map weights(u, v) over all u in $L(x_i)$ and v in $R(x_j)$, it follows from 1a: that it must contain the value of the max-score paths from s to some vertex in $L(x_i)$ and from some vertex in $R(y_j)$ to t that contain at most one super-vertex from each forbidden pair. (All such paths are taken into account in some entry of the map weights(u, v).)

The proofs of 2a:, 2b:, 2c:, and 2d: are nearly identical to those of 1a:, 1b:, 1c:, and 1d:, differing only in reversing the roles of i and j (and replacing the references to Steps 5a and 3 with references to Steps 5b and 4).

Since i and j were arbitrary, it follows that if the predicate M holds for all (i', j') where either $j' < j$ or ($j' = j$ and $i' < i$), then it holds for (i, j) as well. This ends the inductive step.

Together, the base case and inductive step establish that $M(i, j)$ holds for all i and j between 0 and n, where $i < j$, and for $(i, j) = (0, 0)$. This completes the proof.

Now we know that at the end of the algorithm, $Q(i, j)$.weights(u, v).score contains the value of the max-score forbidden-pairs paths from s to u in $L(x_i)$ and from v in $R(y_j)$ to t that contain at most one super-vertex from each forbidden pair (if any such paths exists and $Q(i, j)$.weights(u, v) has no value otherwise). Further, we also know that $Q(i, j)$.weights(u, v).backtrackIndices and $Q(i, j)$.weights(u, v).backtrackKey are also constructed correctly. Specifically, $Q(i, j)$.weights(u, v).backtrackIndices holds indices (i', j') into Q and $Q(i, j)$.weights(u, v).backtrackKey holds a key (u', v') into $Q(i', j')$.weights(u', v') which together possess enough information to move one super-vertex closer in one direction towards the ends of the complete forbidden-pairs path.

Every max-score path from s to t goes from s to a vertex in some $R(x_i)$, across an edge to a vertex in some $L(y_j)$, and then on to t (visiting at most one super-vertex from each forbidden pair). So if we find the score of the max-score forbidden-pairs path for each i and j between 0 and n, the max-score forbidden-pairs path must be among them.

Let i and j between 0 and n be given. If an edge (a, b) from $R(x_i)$ to $L(y_j)$ exists, then the max-score forbidden-pairs path from s to t that contains (a, b) must go from s to some u in $L(x_i)$ to

a to b to some v in $R(y_j)$ to t, and must take the max-score forbidden-pairs paths from s to u and from v to t. Thus its score must be $Q(i,j)$.weights(u,v).score $+ w(u,a) + w(a,b) + w(b,v)$ for some u in $L(x_i)$ and some v in $R(x_j)$. Step 8 computes this value for all u and v for a particular i and j (considering only those u, v where $Q(i,j)$.weights(u,v) exists), and stores the largest such value in $W(i,j)$.pathWeight. Further, Step 8 also stores the key (u,v) which maximizes $W(i,j)$.pathWeight in $W(i,j)$.backtrack. Therefore the loop in Steps 6-8 generates at most $(n+1)^2$ candidate values, one of which must be the score of the max-score forbidden-pairs path. Therefore the largest of the $W(i,j)$.pathWeight must be the desired score, provided some $W(i,j)$.pathWeight has a value.

Given (i^*, j^*), where $W(i^*,j^*)$.pathWeight is the largest pathWeight in W, the backtracking process will construct a max-score forbidden-pairs path from x_0 along some path to x_{i^*} to y_{j^*} along some path to y_0 in the original graph. The proof of this is as follows. The process begins by using $W(i^*,j^*)$.backtrack (called (u^*,v^*)) which is the key in $Q(i^*,j^*)$.weights which maximizes $W(i^*,j^*)$.pathWeight. This key has enough information to begin the construction of the maximally scored forbidden-pairs path P, which will initially be u^*abv^*, where a and b are the vertices in $R(x_{i^*})$ and $L(y_{j^*})$ which connect x_{i^*} to y_{j^*}. The current indices (i,j) are then set to $Q(i^*,j^*)$.weights(u^*,v^*).backtrackIndices, and the current key (u,v) is set to $Q(i^*,j^*)$.weights(u^*,v^*).backtrackKey. At each step of the process, when the current indices are (i,j) and the current key is (u,v), edges are added to P based on $Q(i,j)$.weights(u,v). By the theorem, we know that $Q(i,j)$.weights(u,v) holds backtrackIndices and backtrackKey which maximize the score for the pair of paths from x_0 to x_i and y_j to y_0 where at most one of each forbidden pair is used. Thus, by only adding edges using vertices from backtrackKey, we construct P with the maximal score for paths from x_i to y_j. Therefore once $(0,0)$ is reached, a forbidden-pairs path from x_0 to y_0 has been constructed, with score equal to the largest possible score.

This establishes the correctness of the algorithm.

4 Implementation and Future Work

We have implemented this algorithm as part of a Java implementation of the PepNovo system developed by Frank and Pevzner [3]. Our implementation was reverse-engineered from their description of the scoring algorithm and publically available C++ source code. Their paper states that PepNovo system uses "a dynamic programming algorithm similar to the one due to Chen et al. that is modified to take into account the particulars of our scoring function." We verified, by consulting the PepNovo source code and the PepNovo authors, that the algorithm given here is not the algorithm they used. To the best of our knowledge, the algorithm used in their implementation has not been published elsewhere or had its correctness proved. Although we have finished the implementation of the algorithm given here, we are in the process of finishing our implementation of the PepNovo system. Once completed, we hope to incorporate the implementation into the IBG Desktop, which is a suite of bioinformatics tools, to help in the process of peptide sequencing.

In the course of implementing the PepNovo system, we have come across some problems which could lead to other interesting algorithmic research. Our algorithm finds the longest forbidden-pairs path from the outside of the path and extends it inwards. In other words, we build the forbidden-pairs paths from x_0 and y_0 to some x_i and y_j. While this is correct as a solution of the abstract forbidden-pairs paths with variable vertex scores problem, it has some undesirable consequences for the bioinformatics problem from which the problem arose. In particular, it means that we must start building the path based on the peaks with the lowest intensities in a typical

MS/MS spectrum (i.e., the x_k and y_k peaks for low values of k), which are the peaks we are the least sure represent actual cleavages in the peptide rather than random noise. To compound the problem, mass spectrometer data may not even include these peaks since it often only starts at around 200 Da. Further, even when these low-mass peaks are recorded by the mass spectrometer, there is sometimes ambiguity as to whether these peaks are caused by b- or y-ions, or rather by immonium ions (i.e., a single amino acid from the peptide that has lost a carbon and an oxygen atom).

We believe the effects of the lack of peak information in the low-mass region of the spectrum can be somewhat mitigated by modifying the construction of the spectrum graph. Currently, both PepNovo and our algorithm create edges between vertices if the difference in mass/charge between the peaks that created the vertices is the mass of a single amino acid. If, instead, we create edges between vertices if the mass/charge difference is the mass of one or more amino acids we would not have to be concerned with the absence of low-mass peaks. In fact, it would also have the secondary benefit of allowing us to handle cases where there was insufficient fragmentation of the peptide resulting in gaps in the spectrum corresponding to multiple amino acids' masses.

This technique, however, would not result in a resolution of the ambiguity due to the immonium ions. In fact, it would cause the problem to be exhibited in higher mass regions of the spectrum since there are some cases where distinct amino acid combinations have the same total mass. In general, we do not have a solution to this problem since it seems to be attempting extract too much interpretation out of insufficient data. However, we believe that if a modified version of our algorithm could be discovered that constructs the forbidden-pairs paths from the inside out (starting from the cross over edges and building outwards towards x_0 and y_0), we would be pushing the immonium ambiguity to the ends of the path which could result in a higher confidence in the central portion of the forbidden-pairs path. Based on our inspection of the PepNovo source code, they may in fact have used this approach. However, to the best of our knowledge, this algorithm has never been proven to be correct, leaving the algorithmic problem open.

References

[1] Chen, Ting, et al. "A Dynamic Programming Approach to De Novo Peptide Sequencing via Tandem Mass Spectrometry", *Journal of Computational Biology* Volume 8, Number 3: 325-337, 2001.

[2] Dancik, Vlado, et al. "De Novo Peptide Sequencing via Tandem Mass Spectrometry", *Journal of Computational Biology* Volume 6, Numbers 3/4: 327-342, 1999.

[3] Frank, A. and Pevzner, P. "PepNovo: De Novo Peptide Sequencing via Probabilistic Network Modeling", *Analytical Chemistry* 77: 964-973, 2005.

[4] Kinter, Michael and Sherman, Nicholas E.
Protein Sequencing and Identification Using Tandem Mass Spectrometry. Wiley-Interscience. 2000.

[5] Steele, A. and Angulo, D. "The Illinois Bio-Grid: A Prototype for Industry-Academe Collaboration", *Proceedings of the 2003 Midwest Software Engineering Conference.*