Generating Peptide Candidates from Amino-Acid Sequence Databases for Protein Identification via Mass Spectrometry

Nathan Edwards and Ross Lippert

Celera Genomics, 45 West Gude Drive, Rockville, MD
{Nathan.Edwards,Ross.Lippert}@Celera.Com

Abstract. Protein identification via mass spectrometry forms the foundation of high-throughput proteomics. Tandem mass spectrometry, when applied to a complex mixture of peptides, selects and fragments each peptide to reveal its amino-acid sequence structure. The successful analysis of such an experiment typically relies on amino-acid sequence databases to provide a set of biologically relevant peptides to examine. A key subproblem, then, for amino-acid sequence database search engines that analyze tandem mass spectra is to efficiently generate all the peptide candidates from a sequence database with mass equal to one of a large set of observed peptide masses. We demonstrate that to solve the problem efficiently, we must deal with substring redundancy in the amino-acid sequence database and focus our attention on looking up the observed peptide masses quickly. We show that it is possible, with some preprocessing and memory overhead, to solve the peptide candidate generation problem in time asymptotically proportional to the size of the sequence database and the number of peptide candidates output.

1 Introduction

Reliable methods for identifying proteins form the foundation of proteomics, the large scale study of expressed proteins. Of the available technologies for protein identification, mass spectrometry is the most suitable for high-throughput automation and analysis.

Tandem mass spectrometry, in which peptides are selected and fragmented, reveals peptides' amino-acid sequence structure. In a typical high-throughput setting, a complex mixture of unknown proteins is cut into peptides using a digestion enzyme such as trypsin; fractionated into reduced complexity samples on the basis of some physical or chemical property such as hydrophobicity; and then a tandem mass spectrum is taken for all the observed peptides in each fraction. The end result of such an experiment is a set of a few hundred to a few thousand tandem mass spectra, each of which represents a peptide of about 6-20 amino acid residues. Typically, amino-acid sequences of 8-10 residues carry sufficient information content to determine the protein from which the peptide is derived, at least up to significant homology. This experimental protocol can

R. Guigó and D. Gusfield (Eds.): WABI 2002, LNCS 2452, pp. 68–81, 2002.

reliably identify hundreds of proteins from a complex mixture in a couple of hours of instrument time.

The traditional approach to automated protein identification using mass spectrometry is not nearly as suited to a high-throughput environment. Peptide mass fingerprinting, see Pappin, Hojrup, and Bleasby [11], James et. al. [8], Cottrell and Sutton [4], and Pappin [10], considers the mass spectrum generated by the (typically tryptic) peptides of a single protein. The mass of each peptide, by itself, carries little information content about the original protein, it is only by the simultaneous observation of the masses of many peptides from a protein that reliable protein identification can be carried out. Furthermore, if peptides from many different proteins are observed in the spectrum, the task of reliably determining which peptide ions belong to the same protein becomes near impossible. This means that significant wet lab work must be done in order to reduce the number of proteins in each fraction to as near to one as possible, which ultimately makes peptide mass fingerprinting unsuitable for high-throughput protein identification.

The analysis of each tandem mass spectrum can be done in a variety of ways. Given a good quality spectrum and a peptide that fragments nicely, the amino-acid sequence of the peptide can often be determined *de novo*, merely by looking at the mass differences between peaks in the spectrum. Many algorithmic approaches have been proposed for *de novo* tandem mass spectrum interpretation, see Taylor and Johnson [14], Dancik et. al. [5], Pevzner, Dancik and Tang [13], and Chen et. al. [2] for some examples. In reality, however, in a high-throughput setting, the number of spectra that can be reliably analyzed *de novo* is small. The most reliable approach for analyzing high-throughput tandem mass spectra for protein identification uses amino-acid sequence databases to suggest peptide candidates which are then ranked according to how well they explain the observed spectrum. A number of papers have been published describing tandem mass spectra database search engines and peptide candidate ranking schemes. Some of these approaches have even been commercialized. The first widely adopted program for tandem mass spectra identification via sequence database search was SEQUEST, based on work by Eng, McCormack and Yates [6]. Another successful commercial product, Mascot, is based on work by Perkins et. al. [12]. Bafna and Edwards [1] published a probability based model for scoring and ranking peptide candidates that could be tuned to the peptide fragmentation propensities of different mass spectrometry technologies. Unfortunately, all of this work focuses primarily on determining the correct peptide candidate from among the possibilities suggested by the sequence database. In this paper, we address the problem at the core of all tandem mass spectrum identification sequence search engines, enumerating all appropriate peptide candidates from an amino-acid sequence database efficiently. For each tandem mass spectrum, we require peptide candidates that match the mass of the peptide that was selected for fragmentation.

2 Generating Peptide Candidates

2.1 Problem Formulation and Notation

Peptide Candidate Generation with Integer Weights [PCG(Int)]. Given a string σ of length n over an alphabet $\mathcal{A}$ of size m; a positive integer mass $\mu(a)$ for each $a \in \mathcal{A}$; and k integer mass queries $M_1, \ldots, M_k$, enumerate all (distinct) pairs (i, ω), where $1 \leq i \leq k$ and ω is a substring of σ, such that

$$\sum_{j=1}^{|\omega|} \mu(\omega_j) = M_i.$$

For convenience, we denote the maximum relevant substring mass by $M_{\max} = \max_i M_i$ and the minimum relevant substring mass by $M_{\min} = \min_i M_i$. Further, since the set of query masses may contain repeated values, we define $R_{\max}$ to be the maximum number of repeats.

While we will concentrate primarily on the integer mass version of the peptide candidate generation problem, we must not lose sight of the real mass version which we must eventually implement. The real mass version of the problem defines real masses for each alphabet symbol and supports real query masses with lower and upper mass tolerances.

Peptide Candidate Generation with Real Weights [PCG(Real)]. Given a string σ of length n over an alphabet $\mathcal{A}$ of size m; a positive real mass $\mu(a)$ for each $a \in \mathcal{A}$; and k positive real mass queries with positive lower and upper tolerances $(M_1, l_1, u_1), \ldots, (M_k, l_k, u_k)$, enumerate all (distinct) pairs (i, ω), where $1 \leq i \leq k$ and ω is a substring of σ, such that

$$M_i - l_i \leq \sum_{j=1}^{|\omega|} \mu(\omega_j) \leq M_i + u_i.$$

We define $M_{\max}$ and $M_{\min}$ to be the minimum and maximum relevant substring mass, $M_{\max} = \max_i(M_i + u_i)$ and $M_{\min} = \min_i(M_i - l_i)$ and define $O_{\max}$ to be the maximum number of $[M_i - l_i, M_i + u_i]$ intervals to overlap any mass.

2.2 Application to Peptide Identification via Tandem Mass Spectrometry

In order to focus our analysis of different algorithms for generating peptide candidates, we first describe some of the issues that need be resolved before a peptide candidate generation technique can be used as part of a peptide identification amino-acid sequence search engine.

First, most amino-acid sequence databases do not consist of a single sequence of amino-acids, they instead contain many distinct amino-acid sequences, representing many proteins. Our peptide candidates should not straddle amino-acid

sequence boundaries. This is easily dealt with by introducing an addition symbol to the alphabet $\mathcal{A}$ with a mass guaranteed to be larger than the biggest query.

We usually require significantly more information about each peptide candidate than is represented by its sequence alone. A peptide candidate's *protein context* minimally consists of a reference, such as an accession number, to the sequence database entry containing it, and its position within the sequence. Often, the protein context also contains flanking residues, for checking enzymatic digest consistency quickly, and taxonomy information, for restricting candidates to a species of interest.

Since our experimental protocol cuts each protein with a digestion enzyme, we usually require our peptide candidates have one or both ends consistent with the putative digest cut sites. Trypsin, very commonly used in mass spectrometry applications, cuts proteins immediately after a lysine (K) or arginine (R) unless the next residue is a proline (P). In order to determine whether a peptide candidate is consistent with trypsin, for example, we must use the protein context to examine the symbols before and after a peptide candidate.

If we will always require that both ends of the peptide candidate be consistent with the enzymatic digest, we can improve the performance of any algorithm for the peptide candidate generation problem significantly. However, in practice, we have observed that a significant number of the tandem mass spectra generated in the high throughput setting do not have enzymatic digest consistent endpoints, and requiring this property of our peptide candidates significantly affects our ability to successfully interpret many spectra. As such, we currently do not impose this constraint at the algorithmic level, instead we filter out candidates without digestion consistent endpoints at candidate generation time. Section 6 deals with this issue further.

Peptide redundancy must be carefully considered too. It does us no good to score the same peptide candidate against a spectrum multiple times, but if a peptide scores well, we must be able list all the proteins that it occurs in. If our algorithm does not explicitly eliminate redundant peptide candidates, we can choose to score them multiple times, or store all scored candidates in some data-structure and only score candidates the first time they are observed. Section 4 deals with this issue further.

Post-translational modifications provide yet another consideration that affects algorithmic implementation decisions. If we permit the online specification of the amino-acid mass table in order to support the generation of peptide candidates with a particular post-translational modification, we must discard any algorithm which precomputes peptide candidate masses, unless we can somehow correct for amino-acids with modified masses.

Modeling post-translational modifications becomes more difficult when we must generate peptide candidates in which some amino-acid symbols have multiple masses. In this setting, we can still use the peptide candidate generation formulation above but we must use additional query masses and do additional checks on the generated candidates. For each tandem mass spectrum, we generate a set of mass queries that correct for the presence of a certain number of

residues being translated to a particular modified amino-acid, and check each returned candidate-query mass pair to ensure that the required number of ambiguous amino-acid symbols is present.

2.3 Implications for Peptide Candidate Generation

We consider amino-acid sequence databases of a few megabytes to a few gigabytes of sequence, over an alphabet of size 20. The alphabet is small enough that we can look up the mass table for each symbol in the alphabet in constant time.

A typical set of tandem mass spectra from a single high throughput run consists of hundreds to thousands of spectra, so we expect hundreds to tens of thousands of distinct query masses. Furthermore, these query masses are typically constrained between 600-3000 Daltons with a 2 Dalton tolerance on acceptable peptide candidates. A 2 Dalton tolerance is typically necessary since mass spectrometers select ions for further fragmentation with much less accuracy than they measure the mass of the same ions, and we must consider any peptide candidate that might have been selected, not only those that have been measured. For our peptide candidate generation problem, this implies that the query masses occupy much of the feasible mass range. The importance of this consideration will become clear in Section 5.

We consider the amino-acid sequence database to be provided offline, and that it may be pre-processed at will, the cost amortized over many identification searches. On the other hand, we assume that the alphabet mass function and the query masses are provided only at runtime and any preprocessing based on weights must be accounted for in the algorithm run time.

3 Simple Algorithms

3.1 Linear Scan

Suppose initially that we have a single query mass, M. Finding all substrings of σ with weight M involves a simple linear scan. The algorithm maintains indices b and e for the beginning and end of the current substring and accumulates the current substring mass in $\widehat{M}$. If the current mass is less than M, e is incremented and $\widehat{M}$ increased. If the current mass is greater than $\widehat{M}$, b is incremented and $\widehat{M}$ decreased. If the current mass equals M, then $\sigma_{b\ldots e}$ is output. See Algorithm 1 for pseudo-code.

This algorithm outputs all peptide candidates that match the query mass in $O(n)$ time. **Linear Scan** requires no preprocessing and requires only the string itself be stored in memory. The recent work of Cieliebak, Erlebach, Lipták, Stoye, and Welzl [3] demonstrates how to generalize **Linear Scan** by creating blocks of contiguous alphabet symbols and changing the pointers b and e in block size increments.

3.2 Sequential Linear Scan

The **Sequential Linear Scan** algorithm for the peptide candidate generation problem merely calls **Linear Scan** for each query mass M_i, $i = 1, \ldots, k$. **Sequential Linear Scan** requires no preprocessing of the string and requires

Algorithm 1 Linear Scan

$b \leftarrow 1$, $e \leftarrow 0$, $\widehat{M} \leftarrow 0$.
while $e < n$ **or** $\widehat{M} \geq M$ **do**
  if $\widehat{M} = M$ **then**
    Output $\sigma_{b \ldots e}$.
  if $\widehat{M} < M$ **and** $e < n$ **then**
    $e \leftarrow e + 1$, $\widehat{M} \leftarrow \widehat{M} + \mu(\sigma_e)$.
  else $\{\widehat{M} \geq M\}$
    $\widehat{M} \leftarrow \widehat{M} - \mu(\sigma_b)$, $b \leftarrow b + 1$.

memory only to store the sequence data and query masses. The algorithm takes $O(nk)$ time to enumerate all peptide candidates that match the k query masses. The algorithm not only outputs redundant peptide candidates, it computes their mass from scratch every time it encounters them. On the other hand, the algorithm can easily keep track of protein context information as it scans the sequence database.

3.3 Simultaneous Linear Scan

Unlike **Sequential Linear Scan**, **Simultaneous Linear Scan** solves the peptide candidate generation problem with a single linear scan of the sequence database. For every peptide candidate start position, we generate all candidates with mass between the smallest and largest query masses, and check, for each one, whether or not there is a query mass that it matches. Assuming positive symbol masses and query masses bounded above by some constant, there are at most L candidates that must be considered at each start position. If we pre-sort the query masses, collapsing identical queries into a list associated with a single table entry, we can look up all queries corresponding to a particular substring mass in $O(R_{\max} \log k)$ time. Therefore, we can crudely estimate the running time of **Simultaneous Linear Scan** as $O(k \log k + nLR_{\max} \log k)$.

Theoretically, while it is possible to construct an input for the peptide candidate generation problem that requires the consideration of this many candidates, such an input will never occur in practice. Assuming typical amino-acid frequencies and a 3000 Dalton maximum query, the average number of candidates that must be considered at each start position, or L_{ave}, is about 27, much less than the worst case L of 54. Table 1(a) provides L_{ave} for a variety of publicly available sequence databases. For the case when enzymatic digest consistent endpoints are required, many fewer candidates need to be generated. Table 1(b) gives L_{ave} for tryptic peptide candidates.

In order to take advantage of this, **Simultaneous Linear Scan** must examine only the peptide candidates with masses up to the maximum query mass $M_{\max}$. Algorithm 2 implements **Simultaneous Linear Scan**, achieving a running time bound of $O(k \log k + nL_{\text{ave}}R_{\max} \log k)$. Clearly, as k gets large, this approach will do much better than **Sequential Linear Scan**.

Table 1. Average (Maximum) number of (a) candidates (b) tryptic candidates per start position for various values of $M_{\max}$.

	Sequence Database	500 Da	1000 Da	2000 Da	3000 Da
(a)	SwissPROT	4.491(8)	9.008(17)	18.007(32)	27.010(48)
	TrEMBL	4.492(8)	9.004(17)	18.006(33)	27.010(49)
	GenPept	4.491(8)	9.004(17)	18.005(33)	27.006(49)
(b)	SwissPROT	1.443(4)	2.046(7)	3.137(12)	4.204(18)
	TrEMBL	1.418(4)	2.022(7)	3.066(14)	4.189(19)
	GenPept	1.427(4)	2.036(7)	3.101(14)	4.195(22)

Algorithm 2 Simultaneous Linear Scan

Sort the query masses $M_1, \ldots, M_k$.
$b \leftarrow 1$, $e \leftarrow 0$, $\widehat{M} \leftarrow 0$.
while $b \leq n$ **do**
  while $\widehat{M} \leq M_{\max}$ **and** $e < n$ **do**
    $e \leftarrow e + 1$, $\widehat{M} \leftarrow \widehat{M} + \mu(\sigma_e)$.
    $Q \leftarrow$ **QueryLookup**$(\widehat{M})$.
    Output $(i, \sigma_{b\ldots e})$ for each $i \in Q$.
  $b \leftarrow b + 1$, $e \leftarrow b - 1$, $\widehat{M} \leftarrow 0$.

Like **Sequential Linear Scan**, **Simultaneous Linear Scan** does no preprocessing of the amino-acid sequence database and requires no additional memory above that required for the sequence and the queries. **Simultaneous Linear Scan** generates redundant peptide candidates and can easily provide protein context for each peptide candidates.

4 Redundant Candidate Elimination

Typical amino-acid sequence databases, particularly those that combine sequences from different sources, contain some entries with identical sequences. Fortunately, these redundant entries can be efficiently eliminated by computing a suitable hash on each sequence. Substring redundancy, however, is not as easy to deal with. When all peptide candidates of mass between 600 and 4000 Daltons are enumerated from the publicly available GenPept amino-acid sequence database, over 60% of the peptide candidates output have the same sequence as a candidate already seen. We will denote the compression in the number of candidates that must be considered when this redundancy is eliminated by the substring density, ρ. Figure 1 plots the substring density of various publicly available amino-acid sequence databases for candidates (and tryptic candidates) with mass between 600 and $M_{\max}$, where $M_{\max}$ varies from 610 to 4000 Daltons. Notice that ρ is quite flat for large $M_{\max}$, suggesting substring density significantly less than 1 is not merely an artifact of the length of the substrings considered.

Notice also that the carefully curated SwissPROT sequence database exhibits much greater substring density than the others. This probably reflects

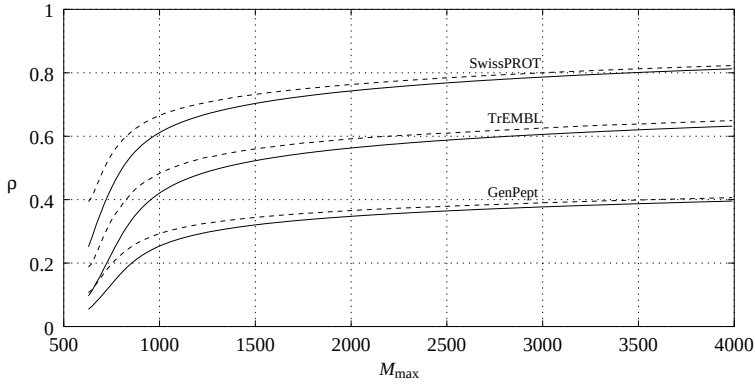

Fig. 1. Substring density of all candidates (solid line) and tryptic candidates (dashed line) with mass between 600 and $M_{\max}$ Daltons.

the curation policy of SwissPROT to collapse polymorphic and variant forms of the same protein to a single entry with feature annotations. If each variant form of a protein was a distinct entry in the database, then its substring density would decrease significantly. On the other hand, GenPept, which contains protein sequence data from many sources and is not hand curated, has much lower substring density. We claim that substring density is not merely inversely proportional to database size, but that it is increasingly difficult to find appropriate ways to collapse similar, but not identical, entries in ever larger sequence databases.

4.1 Suffix Tree Traversal

Suffix trees provide a compact representation of all distinct substrings of a string, effectively eliminating redundant peptide candidates. We discuss the salient properties of suffix trees here, see Gusfield [7] for a comprehensive introduction. Suffix trees can be built in time and space linear in the string length. Once built, all distinct substrings are represented by some path from the root of the suffix tree. Having constructed a suffix tree representation of our sequence database, a depth first traversal of the suffix tree that examines substrings on paths from the root with mass at most the maximum query mass generates all sequence distinct peptide candidates of our sequence database.

As the depth first traversal of the suffix tree proceeds, each candidate substring must be checked to see whether it corresponds to a query mass. As before, we bound the worst case performance of query mass lookup by $O(R_{\max} \log k)$. This results in a running time bound of $O(k \log k + \rho n L_{\text{ave}} R_{\max} \log k)$.

The additional memory overhead of the suffix tree is the only downside of this approach. A naive implementation of the suffix tree data-structure can require as much as $24n$ additional memory for a string of length n. However, as suffix trees have found application in almost every aspect of exact string matching, a great

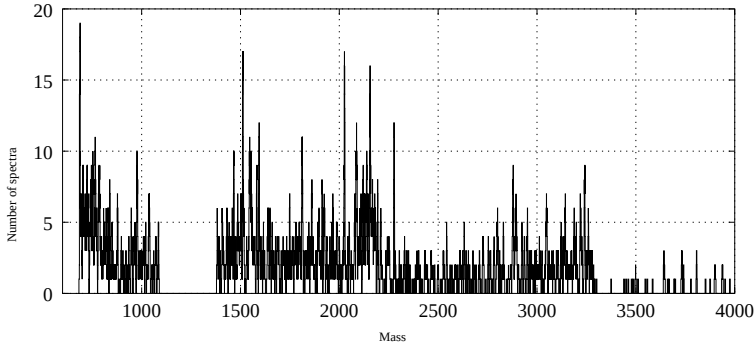

Fig. 2. Overlap plot for 1493 query masses from Finnigan LCQ tandem mass spectrometry experiment.

deal of effort has been made to find more compact representations. For example, Kurtz [9] claims a memory overhead of approximately $8n$ for various biological databases. This degree of compression can also be achieved, in our experience, by using a suffix array data-structure, which represents the same information as a suffix tree, but in a typically more compact form. Again, see Gusfield [7] for an introduction. In theory, a depth first traversal of a suffix array should be quite a bit more expensive than for a suffix tree, but in our experience, the linear scan through the suffix array that is necessary to implement the depth first traversal costs little more than pointer following in the suffix tree.

We note that using a suffix tree traversal for peptide candidate generation can make the protein context of peptide candidates more expensive to compute, particularly for context upstream of the candidate, necessary for determining if the peptide candidate is consistent with an enzymatic digest.

5 Query Lookup

Whether we consider the **Simultaneous Linear Scan** or **Suffix Tree Traversal** algorithm for peptide candidate generation, it should be clear that as the number of distinct query masses k gets large, our running time depends critically on the query mass lookup time. Once k gets too big for the **Sequential Linear Scan** algorithm to be appropriate, effective algorithms for the peptide candidate generation problem must no longer focus on the string scanning subproblem, but instead on the query mass lookup subproblem. Figure 2 plots the degree of overlap of a typical set of query mass intervals derived from a Finnigan LCQ mass spectrometer run containing 1493 tandem mass spectra. Notice the wide range of candidate masses that must be considered by the candidate generation algorithm.

We can obtain an $O(1)$ query lookup time per output peptide candidate by constructing a lookup table, indexed by the set of all possible candidate masses, containing the corresponding query mass indices. This is feasible for

Algorithm 3 Query Mass Lookup Table: Construction: Real Case

$\delta \leftarrow \min_i(l_i + u_i)$.
allocate and **initialize** a lookup table of size $N = \left\lfloor \frac{M_{\max}}{\delta} \right\rfloor + 1$.
for all query masses i **do**
  for all $j = \left\lfloor \frac{M_i - l_i}{\delta} \right\rfloor, \ldots, \left\lfloor \frac{M_i + u_i}{\delta} \right\rfloor$ **do**
    if $j\delta \leq M_i - l_i < (j+1)\delta$ **then**
      $T[j].L \leftarrow i$
    else if $j\delta \leq M_i + u_i < (j+1)\delta$ **then**
      $T[j].U \leftarrow i$
    else $\{M_i - l_i < j\delta$ **and** $M_i + u_i \geq (j+1)\delta\}$
      $T[j].O \leftarrow i$
for all $j = 0, \ldots, N-1$ **do**
  sort the elements i of $T[j].L$ in increasing $M_i - l_i$ order.
  sort the elements i of $T[j].U$ in decreasing $M_i + u_i$ order.

Algorithm 4 Query Mass Lookup Table: Lookup: Real Case

input query mass $\widehat{M}$.
$j \leftarrow \left\lfloor \frac{\widehat{M}}{\delta} \right\rfloor$.
output i for each $i \in T[j].O$.
output i in order from $T[j].L$ until $M_i - l_i > \widehat{M}$.
output i in order from $T[j].U$ until $M_i + u_i < \widehat{M}$.

suitably small integer query masses, as the resulting data-structure will have size $O(M_{\max} + k)$.

We can readily adapt this approach to the real mass case. Algorithm 3 describes the procedure for building the lookup table. Having selected the discretization factor δ for the real masses to match the smallest query mass tolerance and allocating a table of the appropriate size, we iterate through the query masses to populate the table. For each query mass, we determine the relevant table entries that represent intervals that intersect with the query mass tolerance interval. The query mass intervals that intersect the interval $J = [j\delta, (j+1)\delta)$ represented by a table entry j do so in one of three ways: the lower endpoint falls inside J, in which case the query mass index is stored in $T[j].L$; the upper endpoint falls inside J, in which case the query mass index is stored in $T[j].U$; or the query mass interval completely overlaps J, in which case the query mass index is stored in $T[j].O$. The choice of δ ensures that no query mass interval is properly contained in J. Once all the table entries are populated, then for all table entries j, the query mass indices $i \in T[j].L$ are sorted in increasing $M_i - l_i$ order and similarly, the query mass indices $i \in T[j].U$ are sorted in decreasing $M_i + u_i$ order. The resulting lookup table has size bounded above by $O(M_{\max}/\delta + k \max_i(l_i + u_i)/\delta)$ and can be constructed in time bounded by $O(M_{\max}/\delta + k \max_i(l_i + u_i)/\delta + (M_{\max}/\delta) O_{\max} \log O_{\max})$.

For typical values, building this table is quite feasible. $M_{\max}$ is usually between 2000 and 4000 Daltons, while δ is usually about 2 Daltons. Even tens of

thousands of query masses spread over 1000-2000 bins will not cause significant memory or construction time problems. Once built, the cost in memory and construction time is quickly amortized by the $O(1)$ query lookup time.

The bottom line is that it is possible to solve the peptide candidate generation problem (with integer masses) in time $O(M_{\max} + k + \rho n L_{\text{ave}} R_{\max})$ and that it is practical to implement this approach for real query masses. Note that the number of peptide candidates output by any correct algorithm is bounded above by $\rho n L_{\text{ave}} R_{\max}$. This bound is quite tight, particularly when $R_{\max}$ is close to its lower bound of $k/M_{\max}$. Ultimately, this means that we cannot expect to do much better than this running time bound when building the query lookup table is feasible.

6 Enzymatic Digest Considerations

Enzymatic digest considerations offer both a challenge and an opportunity for peptide candidate generation. Enzymatic digest considerations present a challenge, depending on the underlying scan or traversal, because obtaining the necessary protein context in order to check whether a peptide candidate is consistent with a particular enzymatic digest may not be trivial, particularly during a suffix tree based traversal. On the other hand, enzymatic digest considerations present an opportunity, because digest constraints significantly reduce the number of peptide candidates that need to be examined. With careful consideration for the underlying data-structures, it is possible to enumerate peptide candidates that satisfy enzymatic digest constraints at no additional cost, which means that the reduction in the number of candidates output directly impacts the algorithm run-time.

First, we consider how to manage enzymatic digest constraints for the linear scan algorithms. Notice first that since the motifs for enzymatic digestion are typically simple and short, we can find all putative digestion sites in $O(n)$ time. If both ends of the peptide candidates are constrained to be consistent with the enzymatic digest, then the begin and end pointers of Algorithms 1 and 2 can skip from one putative site to the next. Further, if we consider the case when only a small number of missed digestion sites are permitted, we can avoid the generation of candidates with moderate mass, but many missed digestion sites. If only one end of the peptide candidate is required to be consistent with the digest, we must use two passes through the sequence. The first pass constrains the left pointer to putative digest sites, and the second pass constrains the right pointer to putative digest sites and the left pointer to non-digest sites.

Next, we consider how to manage enzymatic digest consistency for the suffix tree based algorithms. Unlike the linear scan algorithms, peptide candidate generation is asymmetric with respect to the ends of the candidate. In fact, there is little we can do within the suffix tree traversal to reduce the amount of work that needs to be done to generate a new peptide candidate in which the right endpoint is consistent with the enzymatic digest. We must still explore all the children of any node we visit, as long as the current peptide candidate mass is

Table 2. total time (s)/number of calls ($\times 10^9$)/time per call ($\times 10^{-6} s$) of the traversal subroutines.

(μsec)	generation	interval search	string scan
Suffix tree	237/1.62/0.146	548/2.90/0.189	115/1.33/0.086
Sim. linear scan	228/2.02/0.112	844/3.53/0.239	87/1.55/0.056

less than the maximum query mass. We can avoid query mass lookups, just as we did for the linear scans, by checking each subsequence with a valid mass for an digest consistent right endpoint before we lookup the query masses. On the other hand, constraining the left endpoint of all candidates to be consistent with the digest can restrict the suffix tree traversal algorithms to a subtree of the suffix tree. In order to accomplish this, we must consider a peptide candidate to consist of the protein context to the left of the candidate prepended to the peptide candidate proper. The non-candidate symbols prepended to the candidate do not contribute to the mass of the candidate, but permit us to look only at those subtrees of the root that begin with a digest consistent motif.

7 Computational Observations

We implemented solutions with a suffix tree and a simultaneous linear scan as described in Algorithm 2. Profiles of these implementations give an estimate to the relative constant factors associated with string traversal versus interval search costs. We ran our implementations against the 1493 sample query masses on the SwissPROT database.

Although we have shown that $O(1)$ lookup of intervals is possible, we have used a skip list to store the intervals in both examples. As the average interval coverage is not large in this example, we thought this would suffice.

Our implementations were compiled in DEC Alpha EV6.7 processors (500 MHz) with the native C++ compiler, `CXX` (with the option `-O4`). Timing was done with `gprof`. We have aggregated the timings for the interval traversal routines, the string traversal routines, and the candidate generation function which calls them alternately.

Table 2 demonstrates that the largest fraction of the cost (by number of calls or time) is spent in the interval lookup, examining about 2 intervals per string search, as Figure 2 suggests. The extra time cost from the redundant string traversals in the simultaneous linear scan implementation is compensated by the speed per-call coming from a simpler scanning routine. The extra interval search time for the simultaneous linear scan is likely due to the increased time spent re-traversing short intervals.

8 Conclusion

We have identified and formulated a key subproblem, the peptide candidate generation problem, that must be solved in order to identify proteins via tandem

mass spectrometry and amino-acid sequence database search. We have outlined the context in which any solution to the peptide candidate generation problem must operate and carefully examined how this context influences the classic algorithmic tradeoffs of time and space.

We have identified a key property of amino-acid sequence databases, substring density, that quantifies the unnecessary peptide candidates output by a linear scan of the sequence database. We have further proposed the use of the suffix array as a compact representation of all substrings of the sequence database that eliminates candidate redundancy.

We have also demonstrated that as the number of query masses increases, query mass lookup time becomes more significant than sequence database scan time. We proposed a constant time query lookup algorithm based on preprocessing the query masses to form a lookup table, and showed that this technique is quite feasible in practice.

Our results for the peptide candidate generation problem depend heavily on our empirical observations about the nature of the problem instances we see every day. We need to obtain better worst case bounds for this problem to make our performance less sensitive to less common extreme instances. We also need to do a comprehensive empirical study to more carefully determine the scenarios in which each of the algorithms identified is the method of choice.

References

1. V. Bafna and N. Edwards. Scope: A probabilistic model for scoring tandem mass spectra against a peptide database. *Bioinformatics*, 17(Suppl. 1):S13–S21, 2001.
2. T. Chen, M. Kao, M. Tepel, J. Rush, and G. Church. A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. In *ACM-SIAM Symposium on Discrete Algorithms*, 2000.
3. M. Cieliebak, T. Erlebach, S. Lipták, J. Stoye, and E. Welzl. Algorithmic complexity of protein identification: Combinatorics of weighted strings. Submitted to Discrete Applied Mathematics special issue on Combinatorics of Searching, Sorting, and Coding., 2002.
4. J. Cottrell and C. Sutton. The identification of electrophoretically separated proteins by peptide mass fingerprinting. *Methods in Molecular Biology*, 61:67–82, 1996.
5. V. Dancik, T. Addona, K. Clauser, J. Vath, and P. Pevzner. De novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 6:327–342, 1999.
6. J. Eng, A. McCormack, and J. Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of American Society of Mass Spectrometry*, 5:976–989, 1994.
7. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
8. P. James, M. Quadroni, E. Carafoli, and G. Gonnet. Protein identification in dna databases by peptide mass fingerprinting. *Protein Science*, 3(8):1347–1350, 1994.
9. S. Kurtz. Reducing the space requirement of suffix trees. *Software–Practice and Experience*, 29(13):1149–1171, 1999.

10. D. Pappin. Peptide mass fingerprinting using maldi-tof mass spectrometry. *Methods in Molecular Biology*, 64:165–173, 1997.
11. D. Pappin, P. Hojrup, and A. Bleasby. Rapid identification of proteins by peptide-mass fingerprinting. *Currents in Biology*, 3(6):327–332, 1993.
12. D. Perkins, D. Pappin, D. Creasy, and J. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567, 1997.
13. P. Pevzner, V. Dancik, and C. Tang. Mutation-tolerant protein identification by mass-spectrometry. In R. Shamir, S. Miyano, S. Istrail, P. Pevzner, and M. Waterman, editors, *International Conference on Computational Molecular Biology (RECOMB)*, pages 231–236. ACM Press, 2000.
14. J. Taylor and R. Johnson. Sequence database searches via *de novo* peptide sequencing by mass spectrometry. *Rapid Communications in Mass Spectrometry*, 11:1067–1075, 1997.