An Automata Approach to Match Gapped Sequence Tags Against Protein Database

Yonghua Han, Bin Ma, and Kaizhong Zhang

Department of Computer Science, University of Western Ontario,
London Ontario, N6A 5B7 Canada
{yhan2, bma, kzhang}@csd.uwo.ca

Abstract. Tandem mass spectrometry (MS/MS) is the most important method for the peptide and protein identification. One approach to interpret the MS/MS data is *de novo* sequencing, which is becoming more and more accurate and important. However *De novo* sequencing usually can only confidently determine partial sequences, while the undetermined parts are represented by "mass gaps". We call such a partially determined sequence a *gapped sequence tag*. When a gapped sequence tag is searched in a database for protein identification, the determined parts should match the database sequence exactly, while each mass gap should match a substring of amino acids whose masses total up to the value of the mass gap. In such a case, the standard string matching algorithm does not work any more. In this paper, we present a new efficient algorithm to find the matches of gapped sequence tags in a protein database.

1 Introduction

Proteins are essential to life, playing key roles in all biological processes. For example, enzymes that catalyze reactions are proteins and antibodies in an immune response are proteins. One of the first steps in understanding proteins is protein identification. Protein identification is to identify the primary structure of a protein, which is a chain of amino acids. There are 20 different amino acids, and therefore, the primary structure of a protein can be represented as a string over an alphabet of size 20. Protein identification is a fundamental problem in Proteomics. Nowadays, tandem mass spectrometry (MS/MS) is becoming the most important and standard technology for this importance protein identification problem [1]. In the current practice of protein identification using MS/MS, purified proteins are digested into short peptides with enzymes like trypsin. Then, tandem mass spectra are measured for the peptides with a tandem mass spectrometer. Fig. 1 shows an example of MS/MS spectrum. A peak in the MS/MS spectrum indicates the mass-to-charge ratio (m/z) of the type of ions that produce the peak, and the intensity of the peak indicates the number of the same type of ions detected. Finally the MS/MS spectra are interpreted by computer software to identify the amino acid sequences of the peptides and proteins.

Many algorithms have been developed and applied in software to interpret the MS/MS data. They can be grouped into two major approaches. The first

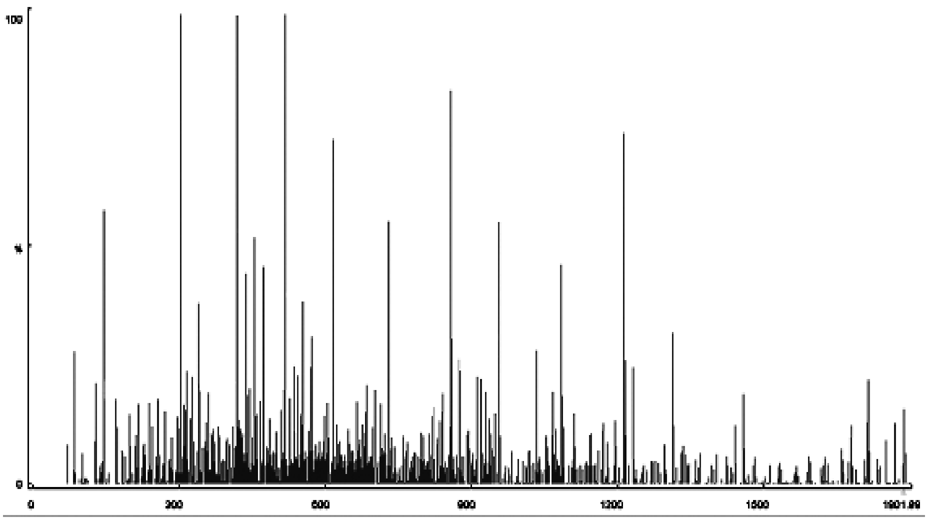

Fig. 1. An MS/MS spectrum

approach correlates MS/MS spectra with peptides in a protein database to find the best matches [10, 18, 19, 25]. We call this approach database search approach. Among the many software programs developed using this approach, Mascot [10] and Sequest [18] are the two most well-known programs.

The other approach is *de novo* sequencing [6, 8, 9, 11, 12, 13, 15, 20, 23, 24], which produces amino acid sequences of the peptides from the MS/MS data directly without the help of the protein database. *De novo* sequencing software often uses the mass difference between the peaks in an MS/MS spectrum to determine the amino acids of the peptide. Because the MS/MS spectra are always not perfect due to the impure sample, incomplete fragmentation and other factors in the MS/MS experiments, the mass difference between two peaks in an MS/MS spectrum may not indicate the mass of only one amino acid. Instead, it may be the sum of the mass of several amino acids. In this case, there can be several combinations of amino acids have the same mass. For example, mass(EE) = mass(GSN) = 258.1 Dalton. Even some single amino acids also have this kind of ambiguous mass, like mass(L) = mass(I) and mass(N) = mass(GG).

As a result, most *de novo* sequencing software cannot determine these ambiguous masses, therefore outputs erroneous results. We call this type of error same mass segments replacement, i.e., one segment of amino acids has the similar mass to another segment. There are some *de novo* sequencing software available, such as commercial software PEAKS [14] and free software Lutefisk [23, 24]. Both PEAKS and Lutefisk have their own mechanisms to reduce the effects of these errors. Lutefisk outputs a mass gap when it cannot confidently determine the sequence that fills the gap. One example output is

```
    [258.1]TLMEYLE[114.0]PK,
```

where the mass gaps [258.1] and [114.0] represent two short segments of amino acids whose masses add up to 258.1 and 114.0 Dalton, respectively. We call such a sequence tag the *gapped sequence tag*.

A possible match of such a tag in the database is EETLMEYLENPK, as follows:

```
Tag:   [258.1]TLMEYLE[114.0]PK
Match: [EE   ]TLMEYLE[N    ]PK,
```

where mass(EE) = 258.1 and mass(N)=114.0.

PEAKS uses different colors to represent different confidences on different amino acids in a sequence. PEAKS version `1.x` outputs the low confidence level ($< 80\%$) amino acids with black color, compared to the other colors (red, green, blue) for higher confidences (95%-100%, 90%-95%, 80-90%) [14]. [1] Here we use brackets for the low confidence level parts. For example, the output [GSN]TLMEYLE[GG]PK indicates that PEAKS is not confident at the two segments GSN and GG. Clearly, this [GSN]TLMEYLE[GG]PK can be converted to Lutefisk's output format by replacing GSN with the mass gap [258.1] and GG with [114.0].

The determined parts of sequence tags produced by Lutefisk and high confident parts of PEAKS are very likely the correct sequences. Therefore, when the gapped sequence tag is searched in a database, we require the determined parts to match exactly. However, the mass gaps should be matched by substrings whose total masses equal to the mass gaps.

2 Related Work

Software programs have been developed for the purpose of searching sequence tags in the database to identify peptides and proteins. MS-BLAST [22] uses a BLAST-like algorithm [4, 5], which first finds a seed in the database, and then extends around the seed attempting to find a match. MS-BLAST can find approximate matches using a homology model. However, MS-BLAST does not accept inputs with mass gaps. When mass gaps are present in a query, MS-BLAST requires the user to find all possible exact matches of the tag, and then it uses all those possibilities as query and searches them all together simultaneously. This may create too many possible exact matches when there are very long mass gaps.

Another software program, OpenSea [21], considers the mass gaps and other *de novo* sequencing errors. But the mass gaps can only be matched by substrings with up to 3 amino acids. Therefore OpenSea does not work for long mass gaps either.

In this paper, we present our approach to match gapped sequence tags against database sequences. Our approach separates the query sequence tags into segments by mass gaps. A segment is a short run of amino acid sequence. We modify

[1] The color scheme has been changed slightly in later versions of the software.

Aho-Corasick automaton to find the exact matches of those segments, and a neat algorithm is used to assemble the exact matches together to get the match of the whole gapped sequence tag. We note that a straightforward assembly will not result into a linear running time as our algorithm does.

3 An Algorithm to Match Sequence Tags

In this section, we first describe our algorithm to match one gapped sequence tag against the protein database. Then by slightly modifications, we extend it to simultaneously match multiple gapped sequence tags against the database.

3.1 Problem Definition

In this section we define our problem more formally. Let Σ be an alphabet of constant size. Each letter a in Σ is associated with a mass $m(a)$. Let $s = s[1]s[2]\dots s[k]$ be a string. $|s| = k$ denotes the length of the string, and $m(s) = \sum_{i=1}^{|s|} m(s[i])$.

A gapped sequence tag P is represented by m substrings, $p_1, p_2, \dots, p_m$, and $m-1$ mass gaps, $M_1, M_2, \dots, M_{m-1}$. We want to find all the strings s in the database, such that, $s = p_1 q_1 p_2 \dots q_{m-1} p_m$, and $m(q_j) = M_j$ for $j = 1, 2, \dots, m-1$. We use the notation $|P|$ to denote the total length of the m substrings, p_1, p_2, …, and p_m.

It is possible that a mass gap is at the beginning of or the end of a gapped tag. In such a case, we simply let p_1 or p_m to be null string that matches every position of the database.

Our idea is to use the standard algorithm for multiple string matching to find every occurrence of every substring p_i; and carefully assemble the occurrences together to get the match of the whole gapped sequence tag. The difficulty is that q_j can have variable lengths. However, as later shown in the paper, the constraint $m(q_j) = M_j$ allows us to do the assembly efficiently.

In the following section, we first briefly review the Aho-Corasick algorithm for multiple string matching.

3.2 Aho-Corasick Algorithm

As the extension of Knuth-Morris-Pratt algorithm [16], the Aho-Corasick algorithm is a widely used algorithm that finds all occurrences of multiple patterns in a text using an automaton in linear time. This algorithm serves as the basis for the UNIX tool fgrep, and has many applications in bioinformatics. For example, the Tandem Repeat Occurrence Locator (TROLL) [2] is an application developed on the basis of Aho-Corasick algorithm to find tandem repeats of pre-selected motifs from DNA sequence. Another application called CHAOS [7] using an algorithm including a simplified version of the Aho-Corasick algorithm to find local alignments, which are used as anchor points to improve the running time of DIALIGN [17], a slow but sensitive multiple-alignment tool.

An Aho-Corasick automaton can be constructed by the following two steps: firstly, construct a trie and goto functions from the multiple patterns to be searched. The goto function maps a pair (s, α), where s is a state and α is an input symbol, into a state or the message fail. Secondly, add failure and output functions. The failure function $f(s)$ maps a state into another state s'. $L(s')$ is the longest proper suffix of $L(s)$ such that $L(s)$ is a prefix of some pattern. An output function out(s) gives the set of patterns matched when entering state s. The time complexity of the construction of Aho-Corasick automaton is linearly proportional to the total length of the patterns. The searching time is linear to the text size, and the out(s) function will take $O(z)$ time, where z is the total number of occurrences of all the patterns. The details of constructing such an automaton and proof of time complexity can be found in [3].

However, the Aho-Corasick algorithm or its variances cannot apply to our problem directly. We need to modify it to allow mass gaps between matched segments.

3.3 Our Algorithm

The alphabet set Σ is 20 amino acid letters. We construct an extended Aho-Corasick automaton such that each state has 20 transitions according the segments in the sequence tag P. This automaton is a deterministic finite automaton, which contains five elements: $(Q, \Sigma, \sigma, q_0, F)$. Q is the set of states. Σ is an input alphabet, i.e., 20 amino acid letters. σ is a transition function: $\sigma : Q \times \Sigma \rightarrow Q$. q_0 is the start state, $q_0 \in Q$. The automaton is in the start state before the run of each protein sequence in the database. F is a set of final states, $F \in Q$. Each segment in the sequence tag will produce a final state while constructing the extended Aho-Corasick automaton. Entering a final state, the automaton will output matched position in the protein sequence and the number of segment in the sequence tag.

There are two special pair of amino acids (I,L) and (K,Q). I and L have the same mass, and K and Q have very similar mass. In most case, it is desired that they are not considered to be mismatch. So we have an option that make I and L have the same transition, K and Q have the same transition in each state according the passed parameters when constructing the automaton. For each protein sequence in the database, we will run this modified Aho-Corasick automaton to find all the segment matches of the sequence tag P.

When a segment match is found, it is attempted to assemble with other segment matches. For the joint of two adjacent segments, we need to compute the mass of the substring between the two occurrences of the two segments. Let T be the database string. Given any two positions i and j, it is possible to compute the total mass of substring $T[i..j]$ in $O(1)$ time by building a accumulated protein mass table while we read each amino acid from the database. Let $acm[i]$ be $m(T[1..i])$. Then $m(T[i..j]) = acm[j] - acm[i-1]$.

However, if we straightforwardly compute the mass value between every occurrence of segment p_i and segment p_{i+1}, the running time will be quadratic. Also, because a gap can be filled by substrings of variable lengths, a linear time algorithm cannot be achieved by trying every possible length of the substring.

The Segments Assembly Procedure. We use a set of queues $\{q_1, q_2, \dots q_m\}$ to keep the matches that can be potentially parts of a whole tag match and discard the matches that do not have such a potential. Each q_i corresponds to the i^{th} segment, p_i, of the gapped sequence tag. For the segment match of the first segment, p_1, we simply add the match to q_1. However, if p_i is matched at database position k for $i > 1$, before we add k to the queue q_i, we check whether there is a match of p_{i-1}, such that the two matches have a mass gap equal to M_{i-1} in between. This can be done by checking the queue q_{i-1}. Let k' be the first element in q_{i-1}. Let $M = m(T[(k' + |p_{i-1}|)..k])$, the mass gap between $k' + |p_{i-1}|$ and k.

There are three cases:

Case 1: $M < M_{i-1}$.
This means the occurrence of p_i at k cannot be joined with the occurrence of p_{i-1} at k'. Moreover, the former cannot be joined with any occurrences of p_{i-1} after k', because we use first-in-first-out queues. Therefore, we can safely discard k.
Case 2: $M > M_{i-1}$.
This means the occurrence of p_i at k cannot be joined with the occurrence of p_{i-1} at k'. Moreover, the latter cannot be joined with any occurrences of p_i after k, because we use first-in-first-out queues. Therefore, we can safely delete the occurrence of p_{i-1} at k' and continue to consider the next element in q_{i-1}.
Case 3: $M = M_{i-1}$.
This means the occurrence of p_i at k can be joined with the occurrence of p_{i-1} at k'. But the latter cannot be joined with any occurrences of p_i after k, because that will make the gap too large. Therefore, we can safely delete k' from q_{i-1} but add k to q_i.

Let i be the segment that matches the current database position k. Let $acm[k]$ be the accumulated mass of the database. Then our assembly process for this match is shown in Figure 2.

Lemma 1. *For any position k in q_i, there is a substring $T[j..k+|p_i|]$, such that $T[j..k+|p_i|]$ matches $p_1M_1...p_{i-1}M_{i-1}p_i$. On the other hand, for every substring $T[j..k+|p_i|]$ that matches $p_1M_1...p_{i-1}M_{i-1}p_i$, k is added into q_i once.*

Proof. We prove by induction on i. Obviously the lemma is true for $i = 1$ because of line 2 of Algorithm Assembly. Now suppose the lemma is true for $i = i_0$. We want to prove it is true for $i = i_0 + 1$.

When k is added into q_i in line 13, we know that k' was in q_{i-1}. By induction, we know that there is j such that $T[j..k' + |p_{i-1}|]$ matches $p_1M_1...p_{i-1}$. Also we know by line 11 that the mass gap between the occurrence of p_{i-1} at k' and the occurrence of p_i at k is equal to M_{i-1}. Therefore, $T[j..k + |p_i|]$ matches $p_1M_1...p_{i-1}M_{i-1}p_i$.

On the other hand, if k is such that $T[j..k+|p_i|]$ matches $p_1M_1...p_{i-1}M_{i-1}p_i$, there must be k' that $T[j..k' + |p_{i-1}|]$ matches $p_1M_1...p_{i-1}$. By induction, k' is added into q_{i-1}, and it is removed only when k was added into q_i in line 13.

```
**Algorithm** Assembly($i, k$)
1   if $i = 1$
2       add $k$ to $q_1$
3   else
4       while $q_{i-1}$ is not empty do
5           $k' \leftarrow$ the first element of $q_{i-1}$
6           massDiff $\leftarrow (acm[k] - acm[k' + |p_{i-1}|])$
7           if massDiff $< M_{i-1}$
8               break;
9           else if massDiff $> M_{i-1}$
10              delete $k'$ from $q_{i-1}$
11          else
12              delete $k'$ from $q_{i-1}$
13              append $k$ to $q_i$
14              break;
15  if $q_m$ is not empty
16      output $k$ as a match of the whole tag
```

Fig. 2. The procedure is called whenever the i^{th} segment, p_i, is matched at the database position k. In the algorithm, $acm[k]$ is the accumulated mass of the database. q_i is a first-in-first-out queue

Lemma 2. *The Assembly procedure will be called $O(z)$ time, where z is the total number of occurrences of the segments in the database. The total time that the Assembly procedure takes is also $O(z)$.*

Proof. Because we only call the Assembly procedure whenever we find a match of a segment, the procedure is called $O(z)$ time.

Lines $1-2$ and $15-16$ of the Assembly procedure will take $O(1)$ time for each invocation of the procedure. And lines $5-14$ will take $O(1)$ time for each iteration of the while loop. Therefore, we only need to prove that the while loop is repeated $O(z)$ time in total.

We note that there is one element deleted from the queues every time the while loop is repeated, or the while will break in line 14. And we add at most one element into the queue when a match of a segment is found. Therefore, at most z elements will be added to the queues, and the total time of deletions can be at most z. Therefore, the while loop is repeated in total $O(z)$ time.

Theorem 1. *By using the automaton described above and calling the Assembly procedure whenever a segment match is found, all the matches of a gapped sequence tag can be found in $O(n + |P| + z)$ time, where n is the database size, $|P|$ is the total length of the segments in the queries, and z is the number of occurrences of the segments in the database.*

Proof. The correctness directly follows Lemma 1. For the time complexity, there are three main parts. First, constructing of the extended-AC automaton will take $O(|P|)$ time. Secondly, running the automaton against each amino acid letter in the protein database will take $O(n)$ time. Thirdly, assembling segments when a segment match is found in the database. From Lemma 2, the Assembly procedure will take $O(z)$ time in total.

Multiple Sequence Tags Match. The algorithm discussed above matches a mass gapped sequence tag against the database. It can be modified to adapt to multiple sequence tags. As mentioned in the first section of this paper, in an MS/MS experiment, a protein is digested into many peptides and each peptide will produce a corresponding MS/MS spectrum. The advantage of using multiple sequences is clear: even if some of the sequence tags produced by de novo sequencing software are wrong, the correct and partially correct sequence tags can still provide enough information to identify the protein. To adapt to multiple sequence tags, we need to the following modifications on our previous algorithm:

1. First, let the automaton include all the segments from all the sequence tags. That is, let $\mathcal{P} = \{P_1, P_2, \ldots, P_K\}$ be the set of input query sequence tags from *de novo* sequencing software. Then we include all the segments of all the P_i in the construction of the extended Aho-Corasick automaton. For example, if the query $\mathcal{P}$ contains three sequence tags: HGTVVLTALG[170.10]LK, [184.12]ELFR and [276.14]EFLSD[184.12]LHVLHSK. The automaton is constructed to search the sequence segments {HGTVVLTALG, LK, ELFR, EFLSD, LHVLFSK}.
2. The second modification is output function of the extended AC automaton. We have to modify the output to include the information of which sequence

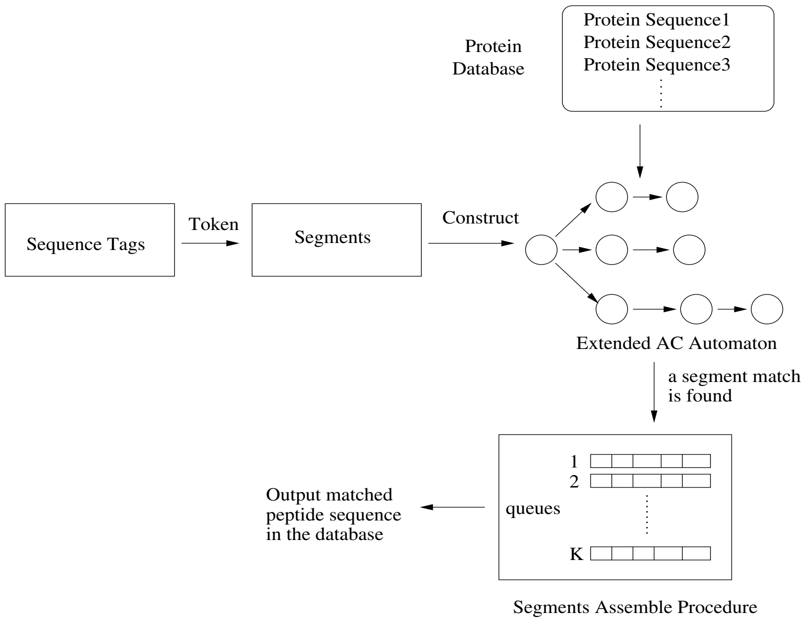

Fig. 3. The process overview of matching gapped sequence tags against a database

tag in the query and which segment in this sequence tag is matched as well as the segment match location in a protein sequence.

3. Third modification is about the queues implemented in segments assembly procedure. Instead of using one set of queues, we need K sets of queues. That is, $q_{i,j}$ corresponds to the j^{th} segment of P_i. And whenever the j^{th} segment of P_i is matched in the database position k, $q_{i,j-1}$ will be checked to determine whether k is inserted to $q_{i,j}$.

The rest of algorithm stays the same as before. It is easy to see the time complexity of matching multiple sequence tags is $O(n + |\mathcal{P}| + z)$, where $|\mathcal{P}|$ is the total length of the sequence segments in all $P_i \in \mathcal{P}$. The process is illustrated in Figure 3.

4 Experiments

We have implemented our algorithm into a Java program. We computed the *de novo* sequencing results of 54 Q-Tof MS/MS spectra, using PEAKS and Lutefisk, respectively. The 54 MS/MS spectra were obtained from the paper [14] of Ma *et al* and can be found at www.csd.uwo.ca/~bma/peaks. Both PEAKS and Lutefisk output some sequence tags completely correct and some sequences only partially correct. For example, PEAKS output [NGG]PVPKPK and Lutefisk output [228.11]PVPKPK while the correct peptide sequence is DIPVPKPK. The brackets in the PEAKS output indicates low confident level (lower than %80). We submitted each of the sequences to our program to search in Swiss-Prot protein database and examined whether the first match output by our program is the correct sequence. Lutefisk only outputs gapped sequence tags when it is not confident to some amino acids. PEAKS' output is also converted to gapped sequence tags by using the method discussed in the introduction.

Table 1 show the difference on the numbers (ratios) of correctly computed sequences before and after using our program on the two sets of data.

Table 1. A comparison of peptide identification before and after using our program

	PEAKS	Lutefisk
Before our program	22 (41%)	11 (20%)
After our program	36 (67%)	23 (42%)

A more interesting use of our program is protein identification with multiple sequence tags. The *de novo* sequencing results of the spectra for the same protein can be submitted to our programm together. The 26 spectra we used are in four groups, each for one of the four proteins, beta casein (bovine), myoglobin (horse), albumin (bovine), and cytochrome C (horse). By submitting each group of PEAKS (or Lutefisks) *de novo* sequencing results to our program, all of the four proteins can be correctly identified. The organisms of the proteins can also be identified except that beta casein (bovine) obtained the same score as beta casein (water buffalo). The reason is that the two proteins differ at only three amino acids, which are not covered by the peptides of the MS/MS spectra.

References

1. Aebersold, R.; Mann, M. “Mass spectrometry-based proteomics”, *Nature* **2003, 422**, 198-207.
2. Adalberto T Castelo, Wellington Martins, Guang R. Gao, “TROLL-Tandem Repeat Occurrence Locator”, *Bioinformatics* **2002, 18**:634-636.
3. Aho, V.A.; Corasick, J.M. ”Efficient string matching: An aid to bibliographic search”, *Communications of the ACM* **1975 18(6)**:333-340.
4. Altschul S.F. *et al.* “Basic local alignment search tool”, *J. Mol. Biol.* **1990, 215**, 403–410.
5. Altschul, S.F. at. al. “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”, *Nucleic Acids Res.* **1997, 25**, 3389-3402.
6. Bartels, C. “Fast algorithm for peptide sequencing by mass spectroscopy”, *Biomed. Environ. Mass Spectrom.* **1990, 19**, 363-368.
7. Brudno, M. *et at.* “Fast and sensitive multiple alignment of large genomic sequences” *BMC Bioinformatics* **2003** 4:66.
8. Chen, T. *et al.* “A dynamic programming approach to *de novo* peptide sequencing via tandem mass spectrometry”, *J. Comp. Biology* **2001, 8(3)**, 325-337.
9. Dančík, V. *et al.* “*De novo* protein sequencing via tandem mass-spectrometry”, *J. Comp. Biology* **1999, 6**, 327-341.
10. Eng, J.K.; McCormack, A.L.; Yates, J.R. “An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database”, *J. Am. Soc. Mass Spectrom.* **1994, 5**, 976-989.
11. Fernández-de-Cossío, J. at. al. “Automated interpretation of high-energy collision-induced dissociation spectra of singly-protonated peptides by “SeqMS”, a software aid for *de novo* sequencing by MS/MS”, *Rapid Commun. Mass Spectrom.* **1998, 12**, 1867-1878.
12. Hines, W.M. *et al.* “Pattern-based algorithm for peptide sequencing from tandem high energy collision-induced dissociation mass spectra”, *J. Am. Sco. Mass. Spectrom.* **1992, 3**, 326-336.
13. Ma, B.; Tromp, J.; Li M. “PatternHunter: faster and more sensitive homology search”, *Bioinformatics* **2002, 18(3)**, 440-445.
14. Ma B. *et al.* “PEAKS: powerful software for peptide *de novo* sequencing by MS/MS”, *Commun. Mass Spectrom.* **2003, 17(20)**, 2337-2342.
15. Ma, B.; Zhang, K.; Liang C. “An effective algorithm for the peptide *de novo* sequencing from MS/MS spectrum”, *CPM’03*, **2003**, 266-278.
16. Morris, J.H.; Pratt. V.P. “A linear pattern-matching algorithm” 1970 Report 40, University of California, Berkeley.
17. Morgenstern B “DIALIGN 2: improvement of the segment-tosegment approach to multiple sequence alignment” *Bioinformatics* **1999, 15**:211-218.
18. Perkins, D.N. *et al.* “Probability-based protein identification by searching sequence database using mass spectrometry data”, *Electrophoresis* **1999, 20**, 3551-3567.
19. Pevzner, P.A.; Dančík, V.; Tang, C. “Mutation tolerant protein identification by mass spectrometry”, *J. Comp. Biology* **2000, 6**, 777-787.
20. Sakurai, T. *et al.* “Paas3: A computer program to determine probable sequence of peptides from mass spectrometric data” *Biomed. Mass spectrum.* **1984, 11(8)**, 396-399.

21. Searle, B.C. *et al.* “High-throughput identification of proteins and unanticipated sequence modifications using a mass-based alignment algorithm for MS/MS *de Novo* sequencing results”, to appear in *Anal. Chem.*.
22. Shevchenko, A. *et al.* “Charting the proteomes of organisms with unsequenced genomes by MALDI-quadrupole time-of-flight mass spectrometry and BLAST homology searching”, *Anal Chem.* **2001, 73(9)**, 1917-26
23. Taylor, J.A.; Johnson, R.S. “Sequence database searches via *de novo* peptide sequencing by tandem mass spectrometry”, *Rapid Commun. Mass Spectrom.* **1997, 11**, 1067-1075.
24. Taylor, J.A.; Johnson, RS. “Implementation and uses of automated *de novo* peptide sequencing by tandem mass spectrometry”, *Anal. Chem.* **2001, 73**, 2594 - 2604.
25. Yates, J.R.I. *et al.* “Method to correlate tandem mass spectra of modified peptides to amino acid sequences in the protein database”, *Anal. Chem.* **1995, 67**, 1426-36.