RAPID COMMUNICATIONS IN MASS SPECTROMETRY
*Rapid Commun. Mass Spectrom.* 2010; **24**: 807–814
Published online in Wiley InterScience (www.interscience.wiley.com) DOI: 10.1002/rcm.4448

RCM

# Speeding up tandem mass spectrometry based database searching by peptide and spectrum indexing

**You Li[1,2†], Hao Chi[1,2†], Le-Heng Wang[1], Hai-Peng Wang[1,2], Yan Fu[1], Zuo-Fei Yuan[1,2], Su-Jun Li[3], Yan-Sheng Liu[3], Rui-Xiang Sun[1], Rong Zeng[3] and Si-Min He[1*]**

[1]Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2]Graduate University of Chinese Academy of Sciences, Beijing 100049, China
[3]Research Center for Proteome Analysis, Key Lab of Proteomics, Institute of Biochemistry and Cell Biology, Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences, Shanghai 200031, China

Received 11 November 2009; Revised 7 January 2010; Accepted 8 January 2010

**Database searching is the technique of choice for shotgun proteomics, and to date much research effort has been spent on improving its effectiveness. However, database searching faces a serious challenge of efficiency, considering the large numbers of mass spectra and the ever fast increase in peptide databases resulting from genome translations, enzymatic digestions, and post-translational modifications. In this study, we conducted systematic research on speeding up database search engines for protein identification and illustrate the key points with the specific design of the pFind 2.1 search engine as a running example. Firstly, by constructing peptide indexes, pFind achieves a speedup of two to three compared with that without peptide indexes. Secondly, by constructing indexes for observed precursor and fragment ions, pFind achieves another speedup of two. As a result, pFind compares very favorably with predominant search engines such as Mascot, SEQUEST and X!Tandem. Copyright © 2010 John Wiley & Sons, Ltd.**

High-throughput protein identification is the basis of proteomics, with the tandem mass spectrometry (MS/MS)-based shotgun approach as the technique of choice. Among the various approaches to data analysis, database search engines have been most reliably and most widely used, such as Mascot,[1] SEQUEST,[2] pFind,[3–5] X!Tandem,[6] OMSSA,[7] and Phenyx.[8] While the majority of research efforts have aimed to improve effectiveness by the design of new scoring and validating algorithms, database search engines are facing a serious challenge of efficiency, owing to the following reasons.

Firstly, the size of protein sequence databases is ever increasing. From IPI.Human.v3.22 to IPI.Human.v3.49, the count of the protein sequences has increased by nearly a third. Besides, along with the evolution and even revolution of genome sequencing technologies, proteogenomic research hopes to use genome translated protein sequences for protein identification. As an example, the EST database (Human.12.06) will be translated into 8,163,883 protein sequences, over 100 times larger than the human proteome IPI.Human.v3.49, which has only 74,017 protein sequences.

Secondly, increasing demand for consideration of semi- or non-specific digestion results in 10 or 100 times more digested peptides respectively than specific digestion, as shown in Table 1.

Thirdly, post-translational modifications (PTMs) produce exponentially more modified peptides. At present over 500 types of PTMs are recorded in the Unimod database.[9] If we select ten common variable PTMs and restrict the number of modification sites in a peptide to a maximum of five, the number of tryptic peptides of the human proteome will be expanded over 1000 times, as shown in Table 2.

Although the performance of computing hardware is improving steadily, it cannot catch up with the expansion pace of candidate peptides. In the meantime, a mass spectrometer like the LTQ generates about five tandem mass spectra per second, or about 400,000 spectra per day, and the generation speed would surely increase steadily. All these factors contribute to the serious challenge of efficiency for protein identification search engines.

Recent years have witnessed various attempts to improve the identification efficiency. The most noticeable is the peptide sequence tag approach, pioneered by Mann and Wilm,[10] and followed by GutenTag,[11] MultiTag,[12] InsPecT,[13] ByOnic,[14] and Spectral Dictionary.[15] However, owing to the limitations of spectra resolution and accuracy, peptide sequence length, and charge states, automatic and reliable extraction of a peptide sequence tag or tags from a tandem mass spectrum is far more than a trivial task, and hence this approach is not as widely used as the classical database search engines.

There have also been several attempts to improve the design of classical database search engines. For example, Edwards and Lippert considered the problem of generating the peptides from protein sequences and matching with spectra,[16] Tang and co-workers intended to use peptide and b/y ions indexes,[17] Yen *et al*. developed a method to remove

*Correspondence to*: S.-M. He, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China.
E-mail: smhe@jdl.ac.cn
†These authors contributed equally to this work.

RCM

**Table 1.** The scale of peptide sequences under tryptic digestion

| Database | Yeast v.20080606 | IPI-Humanv.3.49 | Swiss-Prot v.56.2 |
|---|---|---|---|
| Proteins | 6,717 | 74,017 | 398,181 |
| Fully specific peptides | 741,476 | 7,412,821 | 34,764,218 |
| Semi-specific peptides | 11,659,540 | 119,136,531 | 556,602,826 |
| Non-specific peptides | 120,464,808 | 1,230,715,950 | 5,605,491,572 |

Note: peptide mass: 800–6000 Da, peptide length: 4–100 amino acids, non-specifically digested peptide length: 4–50 amino acids, max missed cleavage sites: 2.

unlikely sequences in the database using peptide properties,[18] Dutta and Chen adopted the nearest neighbor search to speed up peptide-spectrum matching,[19] Roos *et al*. proposed to use hardware cache to speed up identification,[20] and Park *et al*. used a peptide index to increase the efficiency of querying candidate peptides for the mass spectrum.[21] However, these sporadic research efforts do not investigate the entire workflow of search engines, and few practical systems are accompanied.

In this paper, we systematically investigate all the efficiency-related steps in the workflow of protein identification: protein digestion *in silico* → peptide modification → peptide-precursor matching → fragment ion-peak matching, and propose several key questions. First, why, how and at what level to construct protein indexes? Being different from Mascot and X!Tandem, only SEQUEST constructs a peptide index. Second, how to deal with PTMs, and whether to construct a modified peptide index? While SEQUEST constructs such an index, Mascot and X!Tandem do not. Third, how to speed up peptide-spectrum matching? Two problems are involved: one is to speed up the mapping of candidate peptides to precursor mass windows, and the other is to speed up the mapping of theoretical fragment ions of a candidate peptide to observed peaks of a tandem mass spectrum. It is worth pointing out that the peptide-spectrum matching step is the basis of any scoring algorithm, and significantly affects the efficiency; however, few studies are known in the field.

**Table 2.** The number of modified peptides in the IPI-Human v3.49 database

| Max. modification sites | Num. modified peptides |
|---|---|
| 0 | 3,309,085 |
| 1 | 25,197,765 |
| 2 | 133,063,810 |
| 3 | 477,180,661 |
| 4 | 1,361,747,010 |
| 5 | 3,395,725,099 |
| 6 | 7,823,314,004 |
| 7 | 17,606,043,889 |
| 8 | 41,148,061,489 |
| 9 | 99,244,365,518 |

Note: Ten modifications are specified: Oxidation (M), Phosphorylation (S, T, Y), Methylation (K, R), di-Methylation (K, R), tri-Methylation (K) and Acetylation (K).

With the design of pFind 2.1 as a running example, we illustrate how to speed up general protein identification search engines by use of indexing techniques. In the next section we describe how to construct peptide indexes more time- and space-efficiently than SEQUEST 2.7 and show a speedup of two to three times over a scheme without peptide indexing. In addition, database searching with such an algorithm compares very favorably with Mascot 2.1.03, SEQUEST 2.7, and X!Tandem (2008.12.01) on the identification speed. We then go on to describe how to index tandem mass spectra precursors and fragment ions to achieve another speedup of two. In the Discussion section we debate the potential for further improving the efficiency of the first-generation protein identification search engines, including Mascot, SEQUEST, X!Tandem, and pFind, which share the common feature of directly mapping peptides to spectra without extracting any information from the spectra.

## PEPTIDE INDEXING

During the general protein identification process, proteins are digested *in silico* into peptides, which are then matched to spectra. However, proteins may produce redundant peptides, which results in redundant peptide-spectrum matching and scoring. For example, redundant peptides make up more than half in the full tryptic digestion of human proteome IPI.Human v3.49, as shown in Fig. 1. Profiling analysis indicates that with a well-designed search engine, peptide-spectrum matching and scoring takes up a large part of the total time in database searching (see Table 6). In addition, the *on-line* digestion *in silico* also takes much time, especially when the spectra are identified in a number of separate batches; such digestion will have to be repeated. Therefore, *off-line* digestion and storing unique peptides in index files is a promising method to remove the above redundant operations.

In terms of peptide indexing, the first question to be answered is what kind of peptides are to be stored, digested peptides or modified peptides? The peptide index of

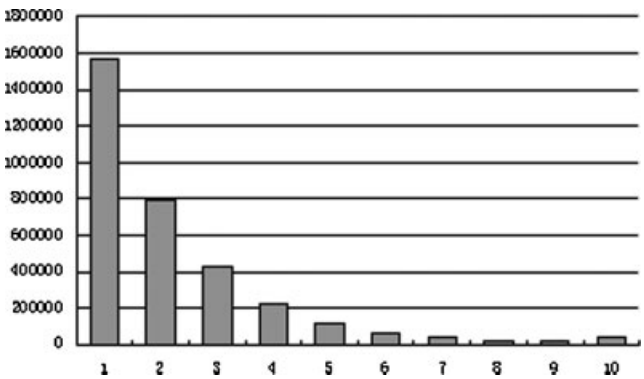


**Figure 1.** The distribution of redundant peptides. The X- axis stands for the peptide count, and the Y-axis stands for the number of peptides with each peptide count. For example, the second column in the histogram means that over 80,000 peptides appear twice.





DOI: 10.1002/rcm

SEQUEST stores all the peptides with PTMs, which suffers from two main shortcomings. Firstly, as indicated by Table 2, the number of modified peptides is huge, and hence the index for the modified peptides will take tremendous time to construct and huge space to store. Secondly, while the set of protein sequences is relatively stable, the set of PTMs specified may be variable for even the same set of spectra; hence the index for the modified peptides has to be reconstructed whenever the specified set of PTMs changes, which is harmful to identification efficiency. In addition, the efficiency of such peptide index construction could be further improved but few studies have aimed at it.

Consequently, pFind constructs *off-line* an index for digested peptides, but generates *on-line* all the modified peptides. Profiling analysis shows that such *on-line* generation of modified peptides only takes less than 5% of total identification time with pFind. Hence this approach has no overhead with the potentially huge space for modified peptides, and little sacrifice of time efficiency. In addition, based on the digested peptides index, other database searching methodologies could easily apply this strategy to generate modified peptides while avoiding redundancy.

The following important questions are concerning index structure, construction and querying. In the following subsection we will talk about the peptide index structure and construction. Since SEQUEST also constructs a peptide index while Mascot and X!Tandem do not, we will compare with SEQUEST in this part. When we talk about the speedup effect of peptide indexing; we compare with not only SEQUEST, but mainly with Mascot and X!Tandem which are more time-efficient.

## Index structure and construction

pFind adopts an inverted index for digested peptides, including the peptide dictionary that stores the distinct peptides and the inverted lists that store the protein IDs that generate the peptides. The data structures for the dictionary and the inverted lists are shown in Table 3. Specifically, pFind uses *Position* and *Length* to represent the peptide sequence, instead of the original sequence like SEQUEST and Interrogator,[17] which decreases the space usage sharply. For example, the average length of tryptic peptides in IPI-Human v3.49 is ~19, which means storing the original sequence will occupy ~19 bytes for each peptide. With the consideration of 20 standard amino acids, each amino acid could be stored in 5 bits, and a peptide would occupy ~12 bytes. However, with the compact representation, pFind only needs exactly 5 bytes. Furthermore, the memory could be used more efficiently and then the time to construct the peptide index would be reduced to a certain extent due to the saving of storage space by such representation.

The construction of the peptide index includes three steps. Firstly, scan the protein sequences, and digest them *in silico* into peptides. Secondly, sort the peptides first by mass and then by sequence at equal mass; as a result, redundant peptides will be placed consecutively, and so it is with the proteins associated with the same peptide. Thirdly, scan the sorted peptides, remove redundant peptides, store the distinct peptides into the dictionary, and put the IDs of all the proteins associated with this peptide into the inverted list, as shown in Fig. 2.

A serious challenge to the construction of an index is that the set of all peptides is too large to be sorted at one time in the memory. Hence a pre-calculation strategy is firstly designed, which calculates the mass distribution of all the peptides first. Given the available memory for index construction, partition the mass range of all the peptides into mass intervals such that the peptides in one mass interval could be sorted in memory (see Supplementary Algorithm 1, Supporting Information).

Because the most time-consuming part is sorting the peptides by sequence, a novel signature method is proposed, using a logarithm Gödel code, which maps each distinct peptide to a distinct floating-point number without any collisions in practice (see Supplementary Algorithm 2, Supporting Information). Sorting the peptides by the signature is five to ten times more efficient than sorting by the sequence, as shown in Table 4. With the pre-calculation strategy and Gödel code, the peptide index can be constructed efficiently in the memory (see Supplementary Algorithm 3, Supporting Information).

Both the construction time and space usage of peptide indexing in pFind are tested and compared with SEQUEST. As is shown in Table 5, pFind is ~5 times faster than SEQUEST under fully tryptic digestion, and ~10 times faster than SEQUEST under semi- and non-specific tryptic digestion, resulting mainly from the following two reasons: Firstly, benefiting from the pre-calculation strategy, pFind constructs the index all in the memory while SEQUEST sorts all peptides in the disk, which is rather slow. The compressed representation of each peptide enables the algorithm to handle more peptides each time, which further speeds up the index construction. Secondly, pFind sorts peptides by their signatures, namely the Gödel codes, instead of the sequences

**Table 3.** Data structure of the peptide dictionary and the inverted list of pFind

| | Field | Type | Annotation |
|---|---|---|---|
| Peptide dictionary | Mass | size _t | Peptide mass |
| | Position | size _t | Position of the peptide in protein database |
| | Length | char | Peptide length |
| | InvertedFilePos | size _t | Pointer to the associated proteins in the inverted list file |
| | Pro_num | size _t | Number of proteins containing this peptide |
| Inverted list | Pro_1_ID_1 | size _t | The first protein containing the first peptide |
| | Pro_1_ID_2 | size _t | The second protein containing the first peptide |
| | ... | ... | ... |
| | Pro_2_ID_1 | size _t | The first protein containing the second peptide |
| | ... | ... | ... |





DOI: 10.1002/rcm

RCM

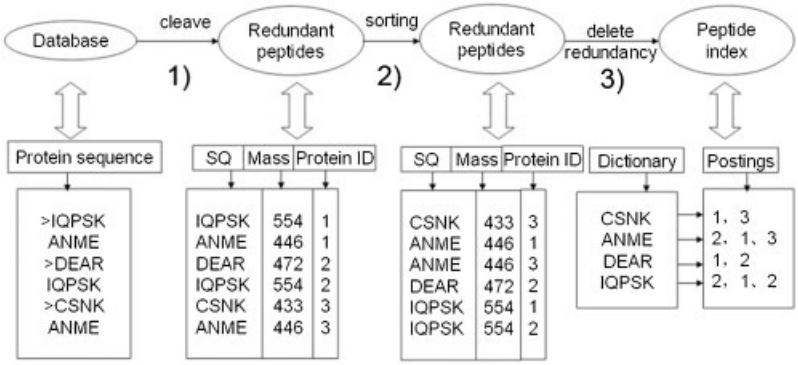


**Figure 2.** The process of constructing the peptide index. (1) Digest protein sequences into peptides. (2) Sort the peptides first by mass and then by sequence; as a result, redundant peptides will be placed consecutively. (3) Remove redundant peptides, store the distinct peptides into the dictionary, and put the IDs of all the proteins into the inverted list.

themselves. On the other hand, even though pFind stores more information in the peptide index than SEQUEST, such as missed cleavages and the number of proteins which generate the peptides, pFind still uses less space than SEQUEST, as is shown in Table 5, especially under semi- and non-specific tryptic digestion. The reason is that pFind compactly represents a peptide sequence by its position and length, not by the sequence itself. (To speed up reading the original peptide sequence, pFind constructs a protein index to represent the FASTA database in a compact style, which is described in detail and compared with Mascot and X!Tandem, that also construct protein indexes, in the Supporting Information.)

**Table 4.** The effect of Gödel code for speeding up the construction of peptide index (in seconds)

| | Yeast | IPI-Human | Swiss-Prot |
|---|---|---|---|
| Sorting the sequence directly | 24 | 334 | 1437 |
| Sorting the Gödel code | **4** | **32** | **275** |

The experiments are under fully specific digestion. Sorting the sequence means, during the index construction, sorting all the peptides by mass and sequence, while sorting the Gödel code means sorting all the peptides by mass and Gödel code.

**Table 5.** Time and space usage of the peptide indexing

| | | Time (s) | | Space (megabytes) | |
|---|---|---|---|---|---|
| Database | Cleavage | pFind | SEQUEST | pFind | SEQUEST |
| Yeast | Fully specific | 4 | 19 | 19 | 21 |
| | Semi-specific | 41 | 766 | 300 | 1,024 |
| | Non-specific | 434 | 3,767 | 3070 | 4,403 |
| IPI-Human | Fully specific | 32 | 122 | 104 | 113 |
| | Semi-specific | 480 | 9,420 | 2,050 | 4,430 |
| | Non-specific | 7,883 | 44,880 | 19,456 | 23,552 |
| Swiss-Prot | Fully specific | 275 | 653 | 600 | 730 |
| | Semi-specific | 5,268 | 51,120 | 11,264 | 39,936 |
| | Non-specific | 81,582 | 407,910 | 107,520 | 122,880 |

## Index query and experiment

Based on the peptide index, the identification workflow is shown in Fig. 3. Three key steps are involved, namely candidate peptide generation, peptide-spectrum matching, and similarity scoring. The next section discusses the latter two steps. The step of candidate peptide generation is to scan the peptide dictionary and, for each unique peptide, generate all of its modified variants as candidates. In order to illustrate the benefit of peptide indexing, the speed of pFind is compared with those of Mascot, SEQUEST and X!Tandem. As indicated before, Mascot only has one workflow, which uses the protein index but not the peptide index. SEQUEST has two workflows, with workflow-1 not using any index and workflow-2 using the peptide index. pFind also has two workflows, with workflow-1 using the protein index only, and workflow-2 using both protein and peptide indexes.

Mascot and SEQUEST only adopt the normal one-step mode of protein identification, which considers all the modifications and the whole protein database through one pass of search. pFind and X!Tandem support both the normal one-step mode and the multi-step mode.[22] In the latter mode, two sets of modifications A and B are specified for two passes of search respectively, where A is usually a subset of B. The first pass considers the whole protein database and the modification set of A, with a small set of proteins identified. The second pass considers the smaller set of proteins identified and the larger modification set of B, with the identification result as the final output.

The database searching parameters are shown in Supplementary Table S3 (see Supporting Information). The MS/MS data in Exp.1 are from a previously reported dataset,[23] and the data in Exp.2 were generated by another liquid chromatography/tandem mass spectrometry (LC/MS/MS) experiment, analyzing a mixture of human serum proteins. Specifically, in the multi-step method of pFind and X!Tandem, phosphorylation is considered only in the second round of database search, which is performed against a much smaller database. In all experiments reported in this the paper, we used a Dell server, which has Intel Xeon (R) 5100 @ 1.60GHz, 4CPU and dual core with 8GB memory. All the experiments mentioned here were done with this computer.





DOI: 10.1002/rcm

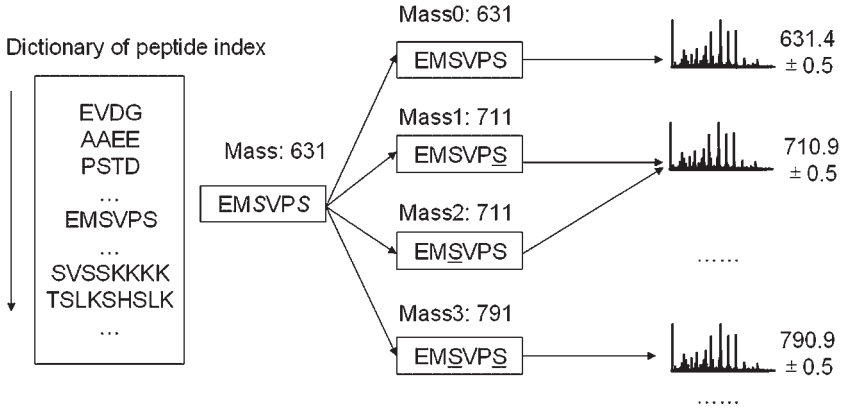


**Figure 3.** In the workflow of pFind, three key steps are involved. Firstly, scan the peptide index and for each peptide, generate candidate modified peptides; secondly, find all spectra whose masses fall in the tolerance window of one peptide; thirdly, score each peptide-spectrum match.

**Table 6.** Time usage distribution and scoring number of pFind in Exp.2

| Time usage | pFind without peptide index | pFind with peptide index |
|---|---|---|
| Candidate peptide generation | 22 min | 15 min |
| Peptide-spectrum matching | 27 min | 17 min |
| Similarity scoring | 1768 min | 815 min |
| Number of scoring | 12,415,099,229 | 6,184,080,825 |

As shown in Table 7, pFind, using the peptide index, performs better than Mascot and X!Tandem in both modes. For example, in the multi-step mode and Exp.2, pFind is more than five times faster than X!Tandem. Since pFind and X!Tandem use similar multi-step search strategies and scoring models, it is reasonable to conclude that the well-designed peptide index is indeed beneficial in database searching. In addition, although pFind adopts a more complicated scoring model[3] than SEQUSET and generates modified peptides on-line, it is still more than ten times faster than SEQUEST, which further confirms the benefit of the peptide indexing strategy proposed in this section. Besides, the identification results are shown in Supplementary Tables S4 and S5 in the Supporting Information.

The speed-up effects of pFind's index mechanism are shown as Tables 6 and 7 in detail. In Table 7, using the peptide index, pFind achieves a speedup of two to three in both one-step mode and multi-step than pFind not using the peptide index. In addition, Table 6 shows scoring number and time usage distribution of pFind in Exp.2. Obviously, the similarity scoring occupies more than 90% of the total time, and, using the peptide index, the scoring number decreases nearly 50%, resulting in a high efficiency and also confirming the analysis described at the beginning of the section.

**Table 7.** Time usage of database searching (minute)

| Mode | Search Engines | Exp. 1 | Exp. 2 |
|---|---|---|---|
| One-step | SEQUEST | 296 | 1302 |
| | Mascot | 43 | 1241 |
| | X!Tandem | 68 | 1223 |
| | pFind | 85 | 1768 |
| | **pFind#** | **37** | **815** |
| Multi-step | X!Tandem | 20 | 613 |
| | pFind | 33 | 244 |
| | **pFind#** | **16** | **94** |

# means using peptide index, otherwise means not using peptide index.

In summary, from the above experiments, it is obvious that that peptide index avoids redundant peptide-spectrum matching and redundant scoring, and thus results in a high efficiency. It is also worth emphasizing that such an index strategy could speed up database searching without loss of accuracy, since it has nothing to do with the scoring algorithm itself but only decreases the redundant operations; therefore, it could be widely used in other database search engines, as well as in combination with other database searching techniques, such as paralleling searching.

## TANDEM MASS SPECTA INDEXING

Peptide-spectrum matching consists of two substeps: (1) mapping candidate peptides to precursors, and (2) mapping theoretical fragment ions of a candidate peptide to observed peaks of a tandem spectrum. The step of scoring is to evaluate the similarity between the peptide and the spectrum based on the matched ions/peaks. While different search engines are featured with their respective similarity scoring schemes, their first two steps are essentially the same. When the search engine is adequately optimized, the peptide generation step takes less than 5% of the total identification time, but the peptide-spectrum matching step takes 40% to 60%.



DOI: 10.1002/rcm

RCM

## Speeding up peptide-precursor matching

Given a peptide $P$, a set of tandem mass spectra, and a specified maximum mass deviation, the peptide-precursor matching is to find those tandem mass spectra whose precursors are within the maximum mass deviation from the mass of $P$. Without loss of generality, we check the average times of *failed comparisons* to evaluate algorithms' performance in peptide-precursor matching (it is also used in the next subsection). *Failed comparisons* are defined as unnecessary peak comparisons during the peptide-precursor match, that is, all comparisons before a valid precursor is found.

A simple method is to traverse all spectra in the set, checking each spectrum to see if it matches the peptide $P$. This method is obviously very time-consuming, with time complexity O($N$), where $N$ is the size of the spectra set, because all spectra have to be checked even if there is only one spectrum matching the peptide. A refined method is to sort the spectra by their precursor mass first, and then use a binary search to find if there is one spectrum matching the peptide. Since the spectra are sorted, if one spectrum is matched, then all matched spectra can easily be obtained. This query approach is implemented in X!Tandem, with time complexity O($\log_2 N$).

However, when the number of the spectra increases, the binary search based method is sometimes not efficient enough. We note the fact that the precursor masses are in a pre-specified range, e.g., 400–2000 $m/z$, or 800–6000 Da. This feature can be used to further optimize the peptide-precursor matching. Specifically, we can index all spectra as follows:

1) Sort all spectra by their precursor masses;
2) According to the actual mass range of all the precursors, create bins with a constant width;
3) For each bin, store their corresponding spectra.

Thus we have constructed an inverted index for precursors (for the details, see Supplementary Algorithm 4, Supporting Information): the dictionary contains all the bins, and the inverted list for each bin records the spectrum IDs whose precursor mass windows overlap with the bin. Such an inverted index is illustrated in Fig. 4. When querying spectra by a peptide, we can use a hash function to quickly find the matched bin and locate all the matched spectra. If the width of the bin is appropriate and a good hash function is used, the first proper spectrum can be found in about O(1) time. As a theoretical prediction, the inverted index can efficiently decrease the number of *failed comparisons*. The latter two methods are both implemented in pFind 2.1, and compared in the section entitled 'Experimental' below.

## Speeding up fragment ion-peak matching

When a peptide is matched to a spectrum, i.e., the peptide mass falls in the precursor mass window of the spectrum, the next operation is to match the theoretical fragment ions of the peptide with the observed fragment peaks of the spectrum, which is the common basis for all similarity scoring computations. Since hundreds of thousands of peptides may match the same spectrum, or more specifically the same set of fragment peaks of this spectrum, the fragment ion-peak matching may significantly affect the identification efficiency.

Given a specified set of $T$ fragment ion types (*e.g.* $b$, $y$, $b^{2+}$ and $y^{2+}$), a peptide $P$ with $L$ amino acids corresponds to a theoretical spectrum with $M \approx TL$ fragment ions. Given a tandem mass spectrum $S$ with $N$ sorted fragment peaks and a specified maximum mass deviation, the fragment ion-peak matching is to find out which of the $M$ theoretical fragment ions are matched, i.e., whether the theoretical fragment ion is within the maximum deviation from the $m/z$ ratio of any of

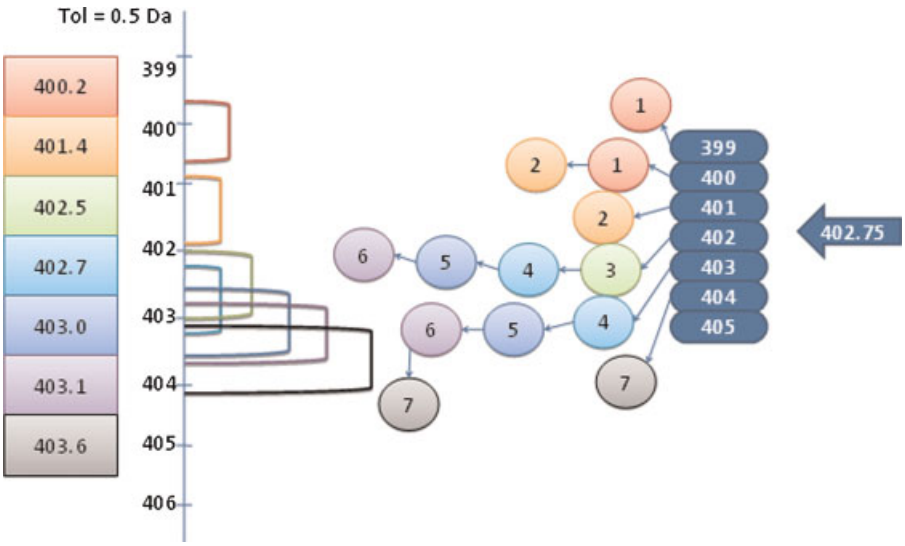


**Figure 4.** An example of inverted index designed for quick peptide-precursor matching. The left part contains several precursors and their mass tolerance intervals according to the tolerance, ±0.5 Da. The right part is the inverted index of the spectra. When a mass $M$ is to be queried, a hash function is used to locate the proper key in the inverted index and all spectra within the tolerance window can be efficiently retrieved.





DOI: 10.1002/rcm

the observed $N$ fragment peaks in the spectrum. This problem is actually the same as the previous peptide-precursor matching problem.

A simple method is to use a binary search in the sorted fragment peaks of the spectrum and find out which of the $M$ theoretical fragment ions are matched. The time complexity of this method is $O(M\log_2 N)$. Another method is to sort the $M$ theoretical fragment ions first, with time complexity $O(M\log_2 M)$, and then the fragment ion-peak matching is equivalent to merging two sorted lists, with complexity $O(M+N)$. According to our experiment, the two methods have similar performance in different data sets, and the latter method has been implemented in the earlier version of pFind, which will be used for comparison in the Experimental section below.

Actually, the fragment ion-peak matching is essentially the same as the peptide-precursor matching, with experimental peaks corresponding to precursors, and theoretical fragment ions corresponding to peptides; hence the indexing technique described in the previous section can be applied here, and it is indeed implemented in pFind 2.1. The fragment peaks dictionary contains a series of bins, and the inverted list for each bin records the experimental peaks whose $m/z$ windows overlap with the bin. As a further optimization, all values of $m/z$ (stored with the type of double-precision float pointing numbers) are multiplied by a constant and rounded to integers, in order to speed up the numerical calculations. This index can be constructed in $O(N)$ time, which is amortized among the matching with hundreds of thousands of peptides and becomes negligible. For a query with $M$ theoretical fragment ions of a peptide, the time complexity of matching with the $N$ fragment peaks of the spectrum is $O(M)$, regardless of $N$.

### Experimental

Experimental settings are identical to those described in the section entitled 'Index query and experiment' above. Table 8 lists the performances of the different peptide-spectrum matching methods with and without spectrum indexing.

As mentioned above, peptide-spectrum matching is very worth speeding up, and comparisons between masses of fragment ions and experimental peaks are especially time-consuming. In Table 8, the number of *failed comparisons* can be even less than 1 when inverted indexes are used, which indicates that appropriate peaks could be found directly or via averaging at most one time to compare the fragment ion mass with masses in the inverted index. Therefore, inverted indexes for precursors and fragment peaks of the tandem mass spectra could sharply reduce the number of the *failed comparisons* in peptide-spectrum matching by as much as ten times, and in consequence the total peptide identification time is reduced significantly. In Exp.2, the identification time is sharply reduced by 47.2%, which means that the spectrum indexing achieves another speedup of two.

**Table 8.** Performance of database searching with and without the spectrum indexing

| | | Number of *failed comparisons* | |
|---|---|---|---|
| | Search time (min) | Peptide-precursor | Fragment ion-peak |
| Exp.1.a | 28 | 10.26 | 5.21 |
| Exp.1.b | 16 | 0.22 | 0.91 |
| Exp.2.a | 178 | 11.20 | 6.36 |
| Exp.2.b | 94 | 0.08 | 0.81 |

Only the performances using multi-step search mode are compared. In Exp.1.a and Exp.2.a no spectrum indexing is used in the database search, while in Exp.1.b and Exp.2.b the spectrum indexing is used.

## DISCUSSION

The key idea of this study is that well-designed indexing technology and query methods can greatly speed up the database search engines for protein identification. Specifically, peptide indexing can achieve a speedup of two to five, and spectrum indexing can achieve a speedup of another two, which has been proved by pFind but is also generally applicable to all other search engines.

While peptide indexing is quite efficient for fully specific digestion, it has potential limitations when dealing with the enormous numbers of peptides from non-specific digestion, since both the peptide dictionary and the inverted lists expand over 100 times in number and in space. Therefore, efficient compression of both the peptide dictionary and the inverted lists is of great importance.

For the compression of the peptide dictionary for non-specific digestion, two features of the candidate peptides might be useful: among the peptides from the same protein sequence, those that share the same start position have consecutive lengths, and those that have the same length start from consecutive positions. Thus a bit-vector can be used to store all these peptides compactly. Another way is to use a suffix tree to store all protein sequences and use a tree-traversal algorithm to generate all possible candidate peptides.[16,24]

It is also necessary to compress the inverted lists. After identification of peptides, the search engine needs to query all corresponding proteins. As the inverted lists are constructed before the peptide identification, it is necessary to store all peptides and their corresponding proteins, no matter whether a peptide occurs in the identification results or not. While a normal identification usually produces 3000–5000 reliable peptides, the number of peptides non-specifically digested *in silico* from Swiss-Prot may go well beyond $10^9$ (see Table 1). Actually only the proteins corresponding to identified peptides need to be queried; this can be accomplished efficiently by the Aho-Corasick algorithm,[25] which accomplishes querying 5000 peptides in a database with about 130,000 proteins in ~20 s on a common personal computer.

The algorithms proposed in this paper can be feasibly applied in multi-core computers and clusters. Generally speaking, spectra are divided into several groups and are identified on each node respectively. Then the peptide index can be constructed and saved on each node. As a further improvement, only the peptides that are in the mass range of a specified spectra group need to be stored in the index files. However, owing to the fast construction of the peptide index, such a segmented index is usually unnecessary. Both





DOI: 10.1002/rcm

multithreads- and cluster-based searching have been implemented in pFind.

In summary, peptide and spectrum indexing are proposed in this paper, which are implemented in pFind and consequently compare very favorably with other predominant first-generation search engines. Indexing technology is a basic method that could be conveniently used in most database search algorithms, since elimination of redundant peptides is always useful before database search. As a result, a peptide index could speed up a database search significantly with little time cost. Furthermore, a spectrum index is also very useful to most scoring schemes. Particularly, the indexing method proposed in this paper has no possibility for decreasing the accuracy, since every distinct peptide is stored in the peptide index. Consequently, it could be also integrated with other related speeding up approaches, including paralleling protein identification, tag-based database searching, spectrum clustering, etc. As a general idea, it is really promising to systematically apply indexing techniques to the design of database search engines for protein identification.

## SUPPORTING INFORMATION

Additional supporting information may be found in the online version of this article.

### Acknowledgements

This work was supported by the National High Technology Research and Development Program (863) of China under Grant Nos. 2007AA02Z315, 2008AA02Z309, the National Key Basic Research & Development Program (973) of China under Grant No.2002CB713807, and the CAS Knowledge Innovation Program under Grant No. KGGX1-YW-13.

## REFERENCES

1. Perkins DN, Pappin DJ, Creasy DM, Cottrell JS. *Electrophoresis* 1999; **20**: 3551.
2. Eng J. *J. Am. Soc. Mass Spectrom.* 1994; **5**: 976.
3. Fu Y, Yang Q, Sun R, Li D, Zeng R, Ling CX, Gao W. *Bioinformatics* 2004; **20**: 1948.
4. Gronert S, Li KH, Horiuchi M. *J. Am. Soc. Mass Spectrom.* 2005; **16**: 1905.
5. Gao Y, Wang Y. *J. Am. Soc. Mass Spectrom.* 2007; **18**: 1973.
6. Craig R, Beavis RC. *Bioinformatics* 2004; **20**: 1466.
7. Geer LY, Markey SP, Kowalak JA, Wagner L, Xu M, Maynard DM, Yang X, Shi W, Bryant SH. *J. Proteome Res.* 2004; **3**: 958.
8. Colinge J, Masselot A, Giron M, Dessingy T, Magnin J. *Proteomics* 2003; **3**: 1454.
9. Available: http://www.unimod.org.
10. Mann M, Wilm M. *Anal. Chem.* 1994; **66**: 4390.
11. Tabb DL, Saraf A, Yates JR III. *Anal. Chem.* 2003; **75**: 6415.
12. Sunyaev S, Liska AJ, Golod A, Shevchenko A. *Anal. Chem.* 2003; **75**: 1307.
13. Tanner S, Shu H, Frank A, Wang LC, Zandi E, Mumby M, Pevzner PA, Bafna V. *Anal. Chem.* 2005; **77**: 4626.
14. Datta R, Bern M. *J. Comput. Biol.* 2009; **16**: 1169.
15. Kim S, Gupta N, Bandeira N, Pevzner PA. *Mol. Cell. Proteomics* 2008; M800103.
16. Bafna V, Edwards N. In. *On de novo interpretation of tandem mass spectra for peptide identification*, RECOMB '03: Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology, 2003; ACM Press: 2003; 9–18.
17. Shilov IV, Seymour SL, Patel AA, Loboda A, Tang WH, Keating SP, Hunter CL, Nuwaysir LM, Schaeffer DA. *Mol. Cell. Proteomics* 2007; **6**: 1638.
18. Yen CY, Russell S, Mendoza AM, Meyer-Arendt K, Sun S, Cios KJ, Ahn NG, Resing KA. *Anal. Chem.* 2006; **78**: 1071.
19. Dutta D, Chen T. *Bioinformatics* 2007; **23**: 612.
20. Roos FF, Jacob R, Grossmann J, Fischer B, Buhmann JM, Gruissem W, Baginsky S, Widmayer P. *Bioinformatics* 2007; **23**: 3016.
21. Park CY, Klammer AA, Kall L, MacCoss MJ, Noble WS. *J. Proteome Res.* 2008; **7**: 3022.
22. Craig R, Beavis RC. *Rapid Commun. Mass Spectrom.* 2003; **17**: 2310.
23. Elias JE, Gygi SP. *Nat. Methods* 2007; **4**: 207.
24. Lu B, Chen T. *J. Comput. Biol.* 2003; **10**: 1.
25. Aho A, Corasick M. *Commun. ACM* 1975; **18**: 333.





DOI: 10.1002/rcm