Indexing and Searching a Mass Spectrometry Database

Søren Besenbacher[1], Benno Schwikowski[2], and Jens Stoye[3]

[1] deCODE Genetics, Reykjavik, Iceland. sorenb@decode.is
[2] Systems Biology Group, Institut Pasteur, Paris, France. benno@pasteur.fr
[3] Technische Fakultät, Universität Bielefeld, Germany.
stoye@techfak.uni-bielefeld.de

Abstract. Database preprocessing in order to create an index often permits considerable speedup in search compared to the iterated query of an unprocessed database. In this paper we apply index-based database lookup to a range search problem that arises in mass spectrometry-based proteomics: given a large collection of sparse integer sets and a sparse query set, find all the sets from the collection that have at least k integers in common with the query set. This problem arises when searching for a mass spectrum in a database of theoretical mass spectra using the *shared peaks count* as similarity measure. The algorithms can easily be modified to use the more advanced *shared peaks intensity* measure instead of the shared peaks count. We introduce three different algorithms solving these problems. We conclude by presenting some experiments using the algorithms on realistic data showing the advantages and disadvantages of the algorithms.

1 Background

Large-scale protein identification methods play a critical role for systems biology approaches [1]. In peptide mass fingerprinting and tandem mass spectrometry, an experimental spectrum is compared to large databases of theoretical spectra in time-consuming linear sweeps [13]. While sequence databases have already been growing exponentially [17], there are a number of recent developments that indicate even stronger growth in the databases of theoretical spectra that need to be searched in unbiased proteomics approaches. These developments include significant increases in the capacity of high-throughput sequencing [11], and the realization that cells abundantly employ post-transcriptional modifications, such as alternative splicing [7], and single-residue modifications [9]. Besides the increase in the size of the search databases itself, ongoing efforts attempt to improve the quality of the scoring functions used to compare an experimental spectrum to a single theoretical spectrum. Typically this comes at the cost of increasing the time of a comparison. Examples of improvements are the prediction of peak intensities in theoretical spectra [3, 5] or the explicit consideration of peptide modifications in tandem mass spectrometry. As a consequence of the above, many large-scale proteomics efforts currently face the problem that the

database searching takes much longer time than the experimental generation of data, making unbiased database search a bottleneck or impossibility in current proteomics pipelines, and interfering with the application of new, sophisticated scoring schemes.

One popular approach to speeding up database searching is to first employ a simple scoring function to filter away spectra that do not score high enough to be considered as true matches. As this step typically allows to quickly exclude most candidate spectra, more sophisticated scoring functions can be applied on the remaining, small, set of spectra. Other approaches attempt to avoid the linear sweep through the database altogether. One such approach is based on a standard method (MVP-tree) for accelerating k-nearest neighbor search in metric spaces [14]. However, the high dimensionality of the spectra significantly limits how much speedup can be achieved by this type of approach. Another approach is based on local sensitivity hashing to obtain fast search times despite the high dimensionality [2]. A drawback of local sensitivity hashing is a non-zero probability that some spectra might be overlooked even though they are within the chosen threshold range of the query spectrum. A third, heuristic approach is based on sequence tags, short sequences of consecutive peaks, and filtering away all spectra that do not fit these tags [4, 10].

The approach presented here is based on the identification of high-scoring spectra according to the simple similarity measures *shared peaks count* (SPC) and its extension *shared peaks intensity* (SPI). Since the same database of theoretical spectra is typically used for many searches, the database can be preprocessed and stored in a data structure that enables faster searching. While in bioinformatics, index-based search has extensively been studied in the context of string pattern matching [12, 15, 16], we are not aware of any such approaches in the context of searching a mass spectrometry database.

In Section 2 we give a formal definition of the search problem, called SPC Range Search Problem, which we consider throughout most of this paper. In Sections 3, 4 and 5 we introduce three algorithms. Section 6 discusses extensions of the basic problem and how our algorithms can be adapted. In Section 7 we present empirical tests of the algorithms on realistic peptide mass fingerprinting data. Section 8 concludes.

2 Problem Definition

In the following, a mass spectrum is represented as a set of integer m/z values in the range $\{1, \ldots, N\}$. A simple similarity measure between two spectra is the *shared peaks count* (SPC), the number of m/z values that two spectra have in common.

Problem 1 (SPC Range Search Problem). Given a set $D = \{T_1, \ldots, T_n\}$ of theoretical spectra $T_i \subseteq \{1, \ldots, N\}$ and a query spectrum $Q \subseteq \{1, \ldots, N\}$, find all the spectra in D that have at least k peaks in common with Q, i.e. identify the set $\{i : SPC(T_i, Q) \geq k\}$ where $SPC(S, Q) := |S \cap Q|$ is the *shared peaks count* of sets S and Q.

Algorithm 1 (Lookup SPC)

Input: array A where $A[x] = \{i \mid x \in T_i\}$ for all $x \in \{1, \ldots, N\}$,
array B where $B[i] = 0$ for all $i \in \{1, \ldots, n\}$
Output: set of indices $I = \{i : |T_i \cap Q| \geq k\}$

```
1: I ← ∅
2: for all x ∈ Q do
3:    for all y ∈ A[x] do
4:       B[y] ← B[y] + 1
5:       if B[y] = k then
6:          I.add(y)
7:       end if
8:    end for
9: end for
```

Let m denote the total number of peaks in all the spectra in the database, i.e. $m = \sum_{i=1}^{n} |T_i|$. If we assume that D and Q are given as sorted lists, then a straightforward algorithm for solving this problem would take $O(\sum_{i=1}^{n}(|T_i| + |Q|)) = O(m + n \cdot |Q|)$ time. However, if we are allowed to build more complex data structures storing D, faster query times are possible. In the following we disregard the preprocessing time needed to build the data structure, as long as it is polynomial, and mainly consider the query times that can be achieved once the data structure is built.

The above formal problem can be applied to the approaches of peptide mass fingerprinting and tandem mass spectrometry. In practice, N is determined by the limited m/z range and the resolution of the instrument used.

3 Lookup Algorithm

A simple data structure for speeding up the query time is an array, A, that maps each integer in the range $\{1, \ldots, N\}$ to a list of the spectra in D that contains the integer in question. While considering the elements of Q one after the other, another array, B, of length n can be used to accumulate the shared peaks count for each of the n spectra. Algorithm 1 shows pseudocode for this algorithm.

3.1 Analysis of Lookup Algorithm

The lookup algorithm assumes its worst case running time if all the peaks of Q occur in all the spectra in the database. In this case the running time is $O(n \cdot |Q|)$. In order to give a better time analysis than this, we have to include some knowledge about the distribution of masses in the spectra in the database so that we know that not all the spectra in the database are expected to be in the result set. Counting the number of times each of the spectra occurs when we look up all the peaks in the query spectrum means that we will never look at the spectra that have no peaks in common with the query spectrum. This should give a substantial speed-up since we expect $|Q|$ to be much smaller than

N. If we assume that none of the N possible peaks is contained in more than $P \cdot n$ spectra, the running time becomes $O(P \cdot n \cdot |Q|)$ which is sub-linear in n since P is a fraction between zero and one. The best case would occur if the peaks in the spectra are uniformly distributed over the range $\{1, \ldots, N\}$. Then the expected value of P would be $\frac{m}{N \cdot n}$, but that is not realistic since we expect to see more small masses than large masses in a spectrum. In experiments with a realistic peptide mass fingerprinting database constructed from a list of all human proteins (see also Section 7) we have measured the value of P to be 0.06 if we only look at masses over 500 Da and use a mass accuracy of 1 Da, see Fig. 1.

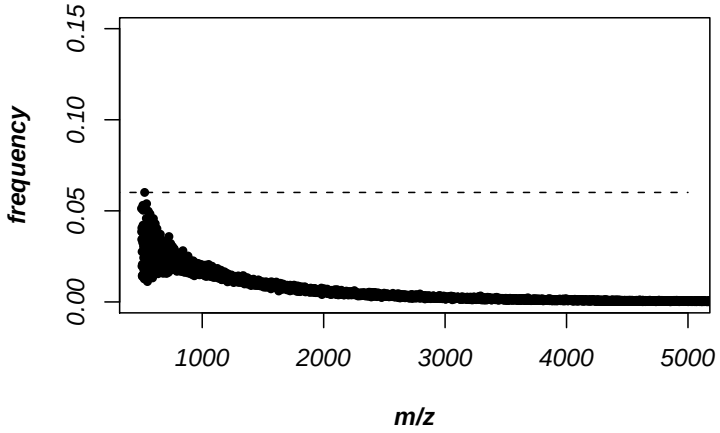

Fig. 1. Distribution of masses in theoretical tryptic digested spectra. Only masses above 500 Da are shown, bin-width is 1 Da. The horizontal line shows the value of P, the maximal frequency of any of the bins.

4 Folding Algorithm

Our second algorithm uses a mapping of the range of masses $\{1, \ldots, N\}$ into $N' \ll N$ bins. The mapping should be defined so that the probability that a spectrum contains a peak belonging to a certain bin should be approximately the same for all bins. One simple possibility that probably satisfies this property is by mapping mass i to bin $H(i) := i \mod N'$.

In the following, let V_S be a vector of length N' so that $V_S[i]$ is the number of peaks in S that fall in the ith bin, $V_S[i] := |\{s \in S \mid H(s) = i\}|$. Given two spectra S and Q, an element $i \in S \setminus Q$ contributes to $V_S[i]$, but not to $V_Q[i]$. Similarly, an element $i \in Q \setminus S$ contributes to $V_Q[i]$, but not to $V_S[i]$. Together, these form the symmetric difference of S and Q, $S \triangle Q := (S \setminus Q) \cup (Q \setminus S)$. Summing over all these elements, we observe that the overall number of element-wise differences

Algorithm 2 (Folding SPC)

Input: $D = \{T_1, \ldots, T_n\}$, and V_T for all $T \in D$
Output: set of indices $I = \{i : |T_i \cap Q| \geq k\}$

```
1: calculate V_Q
2: for i = 1 to n do
3:    if U(T_i, Q) ≥ k then
4:       if SPC(T_i, Q) ≥ k then
5:          I.add(i)
6:       end if
7:    end if
8: end for
```

in all bins, $\sum_{i=0}^{N'-1} |V_S[i] - V_Q[i]|$, is upper-bounded by the cardinality of $S \triangle Q$, $|S \triangle Q| = |S| + |Q| - 2SPC(S, Q)$. Thus, from the two vectors V_S and V_Q we can calculate an upper bound $U(S, Q)$ on the shared peaks count of these two spectra:

$$U(S,Q) \;:=\; \frac{|S| + |Q| - \sum_{i=0}^{N'-1} |V_S[i] - V_Q[i]|}{2} \;\geq\; SPC(S,Q).$$

The idea of the folding algorithm is to preprocess the database by storing for each spectrum $T_i \in D$ the corresponding vector V_{T_i}. Then, for a given query spectrum Q, the upper bounds $U(T_i, Q)$ are computed and subsequently the exact shared peaks count is computed only for those database entries T_i whose bound was larger than k. Algorithm 2 shows pseudocode for this algorithm. A speed-up is achieved if U can be computed faster than SPC and only few computations of the actual shared peaks count are necessary.

4.1 Analysis of Folding Algorithm

Calculating all the upper bounds takes time $O(nN')$, but on top of that we need to calculate the actual shared peaks count of those spectra where the upper bounds were larger than k. To calculate the actual shared peaks count we just use the trivial algorithm for finding the intersection of two sorted lists which takes time proportional to the lengths of the two lists. The number of spectra for which we need to calculate the actual shared peaks count depends on k, N' and Q. In the worst case we would need to calculate the shared peaks count of all the spectra, in which case the running time would be equal to the running time of the straightforward algorithm.

5 Clustering Algorithm

The third algorithm solves the SPC Range Search Problem by dividing the spectra (sets) in D into disjoint subsets $C_1, \ldots, C_\ell$. The main idea is to perform a type of group testing: if the query set Q has less than k peaks in common

Algorithm 3 (Clustering SPC)

Input: clustering $C = \{C_1, \ldots, C_\ell\}$
Output: set of indices $I = \{i : |T_i \cap Q| \geq k\}$

```
 1: I ← ∅
 2: for i = 1 to ℓ do
 3:    peaks ← ∅
 4:    for all x ∈ Q do
 5:       if x ∈ ⋃_{j∈C_i} T_j then
 6:          peaks.add(x)
 7:       end if
 8:    end for
 9:    if peaks.size() ≥ k then
10:       for all j ∈ C_i do
11:          count ← 0
12:          for all x ∈ peaks do
13:             if x ∈ T_j then
14:                count ← count + 1
15:             end if
16:          end for
17:          if count ≥ k then
18:             I.add(j)
19:          end if
20:       end for
21:    end if
22: end for
```

with the union of the sets in a cluster C_i, then none of the spectra in the cluster is in the solution. If however the intersection between Q and the union of the spectra in the cluster is larger than or equal to k, then we need to compare all the spectra in the cluster with the intersection $I_i := Q \cap (\bigcup_{T \in C_i} T)$ in order to obtain $SPC(T, Q) = |T \cap Q| = |T \cap I_i|$. Algorithm 3 shows pseudocode for this algorithm.

The performance of the clustering strategy will depend on how good the clustering is. The number of spectra that end up in clusters that share k peaks with the query spectrum Q should be as small as possible, but at the same time we want there to be as few clusters as possible. The difficulty in making the clustering algorithm effective is in finding a good trade-off between the number of clusters and the probability of the query spectrum having many peaks in common with a cluster.

The probability of a peak from Q belonging to the union of the sets in a cluster should be about the same for all clusters. If we assume that the peaks are uniformly distributed, then this means that the size of a cluster should be some constant factor of the number of possible peaks, i.e. $|\bigcup_{T \in C_i} T| \approx \delta N$ for all i. Finding a clustering of D that satisfies this constraint using as few clusters as possible is an NP-hard problem known as the *Set-Bin-Packing Problem* [6]. For this reason we do not try to find such an optimal clustering but use a

Algorithm 4 (Clustering Heuristic)

Input: $D = \{T_1, \ldots, T_n\}$, δ, N
Output: clustering $C = \{C_1, \ldots, C_\ell\}$ of D

```
 1: i ← 1
 2: while D is not empty do
 3:    C_i ← ∅
 4:    union_i ← ∅
 5:    T ← D.pop()
 6:    C_i.add(T)
 7:    for all x ∈ T do
 8:       union_i.add(x)
 9:    end for
10:    newsize ← |union_i|
11:    while newsize ≤ δN do
12:       best ← D.first()
13:       newsize ← |union_i ∪ best|
14:       for all T ∈ D do
15:          if |union_i ∪ T| < newsize or (|union_i ∪ T| = newsize and |T| > |best|)
             then
16:             best ← T
17:             newsize ← |union_i ∪ best|
18:          end if
19:       end for
20:       if newsize ≤ δN then
21:          C_i.add(best)
22:          for all x ∈ best do
23:             union_i.add(x)
24:          end for
25:          D.remove(best)
26:       end if
27:    end while
28:    i ← i + 1
29: end while
```

heuristic to create the clustering. Algorithm 4 shows pseudocode for an $O(nm)$ greedy heuristic for finding a clustering. The first spectrum in a cluster is picked at random, and afterward we repeatedly add to the cluster the spectrum that increases the union of the number of different peaks in the cluster the least, until the union reaches the limit. If several spectra increase the number of peaks in the cluster by the same amount we add the largest of these to the cluster. Spectra whose size exceeds the limit become singleton clusters.

5.1 Analysis of Clustering Algorithm

For each cluster C_i it takes $O(|Q|)$ time to calculate the intersection I_i if we store the peaks in the clusters in a bitvector so that looking up whether a peak is in the union of a cluster takes constant time. If the size of I_i is k or larger, then

we need to use additional time $O(|I_i| \cdot |C_i|)$. This gives us an expected running time of $O(\ell \cdot |Q| + \sum_{i=1}^{\ell} Pr(|I_i| \geq k) \cdot |I_i| \cdot |C_i|)$. Since we are looking at the expected running time, we can use the expected value $\frac{n}{\ell}$ instead of $|C_i|$ and in view of $|Q| \geq |I_i|$ we can write the expected running time as $O(\ell \cdot |Q| + Pr(|I_i| \geq k) \cdot |Q| \cdot n)$, because we expect $Pr(|I_i| \geq k)$ to be the same no matter what the value of i is. This is always better than the straightforward algorithm since $\ell < n$.

The value of $Pr(|I_i| \geq k)$ depends very much on the value of k. If k is large, the probability will be small and the running time will be dominated by the factor $\ell \cdot |Q|$. If on the other hand k is small, then $Pr(|I_i| \geq k)$ will be large and $Pr(|I_i| \geq k) \cdot |Q| \cdot n$ will dominate the running time.

There is a trade-off between the value of ℓ and $Pr(|I_i| \geq k)$, since making ℓ larger would make the clusters and thus $Pr(|I_i| \geq k)$ smaller. The actual correspondence between $Pr(|I_i| \geq k)$ and ℓ is difficult to calculate since it requires knowledge about the distribution of peaks in the spectra in the database and the query spectrum.

5.2 Recursive Clustering

The clustering idea can be applied recursively to yield a hierarchical clustering. The probability of a peak belonging to a cluster should be the same for all of the clusters on the same level, but it should be smaller for lower levels than for higher levels. We can still use Algorithm 4 to make the clustering, now we just need to apply it recursively to the clusters with a smaller value of δ. The recursive clustering stops when we reach a certain level or if the clusters contain only a few peaks. For a cluster that is a sub-cluster of another cluster we do not need to remember which of all N possible peaks are in the cluster, but only which of the peaks in the super-cluster are in the cluster. This allows us to save some space. For a cluster C let C.subclusters be a list of subclusters of C and let C.rank$[i]$ be x if the ith peak of the super-cluster is in the cluster and there are $x - 1$ peaks in the cluster that have got smaller masses. If the ith peak of the super-cluster is not in the cluster then C.rank$[i]$ should be -1. Algorithm 5 shows pseudocode for a recursive algorithm to search a hierarchical clustering.

6 Extensions

6.1 Shared Peaks Intensities

Apart from their m/z value, peaks in a mass spectrum also have an intensity. Since there is a higher risk that a peak with small intensity does not come from an actual protein fragment, but is just due to random noise, the high intensity peaks should be trusted more than the low intensity peaks. One way of giving more value to high intensity peaks is by using the *shared peaks intensity* (SPI) as similarity measure. The shared peaks intensity of two spectra is the sum of the intensities of all the peaks they have in common. The shared peaks count

Algorithm 5 (Recursive Clustering SPC)

Input: cluster C, list of peaks *peaks*
Output: $I = \{i : |T_i \cap Q| \geq k\}$

```
 1: I ← ∅
 2: newPeaks ← ∅
 3: for all x ∈ peaks do
 4:    if C.rank[x] ≠ −1 then
 5:       newPeaks.append(C.rank[x])
 6:    end if
 7: end for
 8: if newPeaks.size() ≥ k then
 9:    if C contains only a single spectrum T_j then
10:       I.add(j)
11:    else
12:       for all y ∈ C.subclusters do
13:          make recursive call with y and newPeaks
14:       end for
15:    end if
16: end if
```

problem of the previous sections can be seen as a special case of the shared peaks intensity problem where all peaks have intensity one.

Problem 2 (SPI Range Search Problem). Given a set $D = \{T_1, \ldots, T_n\}$ of theoretical spectra $T_i \subseteq \{1, \ldots, N\}$ and a query spectrum $Q = \{q_1, \ldots, q_w\} \subseteq \{1, \ldots, N\}$ and their corresponding intensities $I(q_1), \ldots, I(q_w)$, find all the spectra in D where the sum of the intensities of the peaks they have in common with Q is at least a fraction p of the total intensity of Q, i.e. identify the set $\{i : SPI(T_i, Q) \geq p\}$ where $SPI(S, Q) := \sum_{q \in S \cap Q} I(q) / \sum_{q \in Q} I(q)$ is the *shared peaks intensity* of sets S and Q.

The lookup and clustering algorithms of the previous sections can easily be extended to address this problem instead of the SPC problem. The only change is that instead of counting we now need to sum the intensities.

6.2 Mapping Peaks to Integers

Our algorithms assume that the m/z values are integers even though the values actually produced by the mass spectrometer are not integers. For this reason we need to map the real values to integers. A mass spectrometer will have an associated accuracy stating how much deviation between the real mass and the measured mass can be expected. If we know that there is only a small risk that the measured value deviates more than ϵ from the real value we can map the value x to an integer by first dividing by a value larger than 2ϵ and then rounding down: $x \to \lfloor \frac{x}{r\epsilon} \rfloor$ where $r > 2$. This, however, means that two values less than ϵ apart can be mapped to different integers. One way of addressing this problem

is by mapping the spectra in the database as described above, but mapping the query spectrum so that if there are two integer values that are within $\frac{1}{r}$ of $\frac{x}{r\epsilon}$, then both of these integers are in the query set. Making the integer conversion this way means that one might get a higher shared peaks count than one would get using a more precise alignment between spectra. This is, however, not a big problem since we intend to use our method as a fast filtering where a reasonable number of false positives is acceptable, but false negatives are not. The mapping described above might double the size of the query spectrum which of course affects the running times, but it ensures that there will be no false negatives.

6.3 Tandem Mass Spectrometry

The algorithms could also be used on tandem mass spectrometry data. In tandem mass spectrometry, the query spectra come from fragmented "parent" peptides, whose mass is usually known. This additional information means that, in the case of tandem mass spectrometry, the database does not need to be preprocessed as a whole, but independent index structures can be made for all different parent masses (the masses are rounded to integers). Even though this means that the spectra in the databases are distributed on many separate data structures, it does not mean that the data structures are necessarily small.

In tandem mass spectrometry one often wants to consider not just one spectrum for each peptide but also take into account the possibility that a peptide could have been modified by, for example, phosphorylation. The so called *virtual database* solution of searching for peptide modifications means that, for each possible position of all interesting modifications, a new spectrum is added to the database, resulting in a large increase of the database size. Hopefully the algorithms presented here help making this virtual database approach more feasible.

7 Experiments

We have implemented the following four algorithms in C++ in order to evaluate their search times in a comparative setting:

Simple: the straightforward algorithm that scans the database linearly for each query spectrum.

Lookup: the algorithm that stores for each mass a list of the spectra that contain this mass (Algorithm 1).

Folding: the algorithm that maps the mass range $\{1, \ldots, N\}$ into $N' \ll N$ bins (Algorithm 2).

Cluster: the algorithm that divides the spectra into disjoint subsets and then searches these subsets recursively (Algorithms 3–5).

In order to test the running times on realistic peptide mass fingerprint (PMF) data, we created a PMF database from a list of all human proteins obtained from The International Protein Index [8]. From each protein we generated a theoretical spectrum by simulating a tryptic digest of the protein. Trypsin is an

enzyme that cleaves the protein after each occurrence of the amino acids lysine or arginine, except if the next amino acid is proline. So for the generation of theoretical spectra we first split the proteins into substrings based on the just mentioned rule and then calculated the masses of these substrings by summing the masses of their amino acids and then converting them to integer values.

We have simulated query spectra by randomly drawing a specific number of different peaks between 400 and 5000 Da using the distribution of peaks that we observed in the database. Figure 2 shows query times for searching 1000 different query spectra with 50 peaks each for different values of the threshold parameter k. It can be seen that while the time usage of the simple algorithm and the lookup algorithm is almost unaffected by changes in k, the time usage of the cluster algorithm does depend on this parameter. The lookup algorithm is always faster than the simple algorithm and for small values of k it also beats the clustering algorithm, but for larger values of k the clustering algorithm is the fastest.

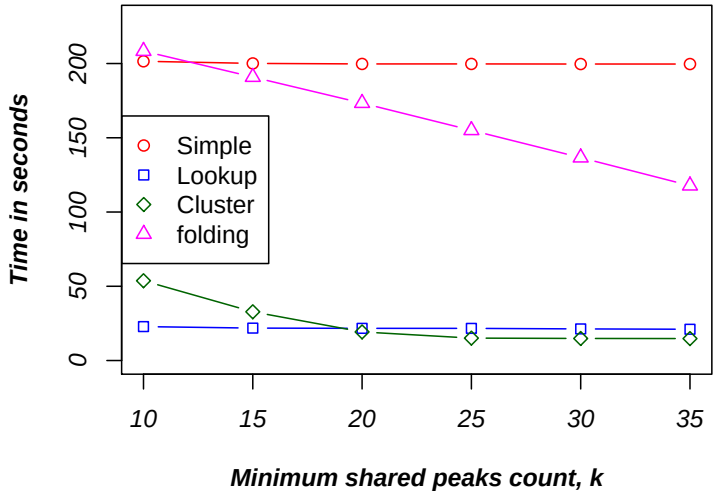

Fig. 2. Experimental results on simulated PMF data. The vertical axis shows the time in seconds it took to perform 1000 queries. The horizontal axis shows different values of the minimum shared peaks count k. The clustering algorithm used a recursive clustering with three levels where the δ parameters for the three levels were 0.05, 0.02 and 0.01.

Figure 3 compares the time usage for the clustering algorithm using clusterings built with different values of δ and a version with recursive clustering (parameters are given in the figure caption). The results show that for a given k the recursive clustering algorithm is not much better than the best of the single level clusterings, but the main effect of the recursive clustering is that the curve is flatter so that the same data structure is good for more values of k.

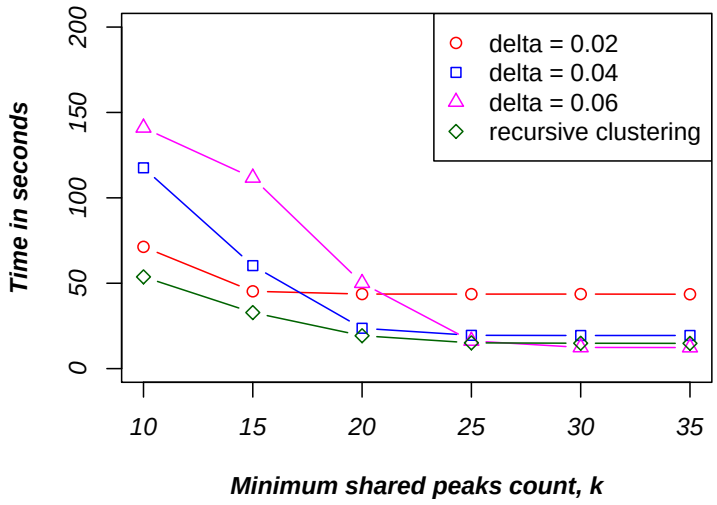

Fig. 3. Comparison of the clustering algorithm with different values of the δ parameter on simulated PMF data. The vertical axis shows the time in seconds it took to perform 1000 queries. The recursive clustering had three levels and the δ parameters for the three levels were 0.05, 0.02 and 0.01.

The International Protein Index we used contains 69164 human proteins. To see how the size of the database affects the running times of the algorithms we also tried creating some smaller databases by only using a fraction of the 69164 proteins and some larger ones by including some extra spectra that were generated by adding a small value to all the peaks of an existing spectrum. Figure 4 shows running times for varying database sizes when searching for query spectra with 50 peaks and a threshold of $k = 15$. It is interesting to note that while the clustering algorithm takes twice the time of the lookup algorithm for $n = 25000$ it only takes half the time for $n = 200000$. This means that the clustering algorithm scales better with database size than the lookup algorithm does.

We have also tested the algorithms on the shared peaks intensity measure by giving the peaks in the query spectra intensities picked randomly between 0 and 1 from a uniform distribution. The results can be seen in Fig. 5. They are similar to the corresponding results for SPC.

8 Conclusion

We have developed three algorithms for searching in an indexed mass spectrometry database using the simple shared peaks count and shared peaks intensity similarity measures. The algorithms can be used to filter potential candidates in a database before ranking them using a more sophisticated scoring thus reduc-

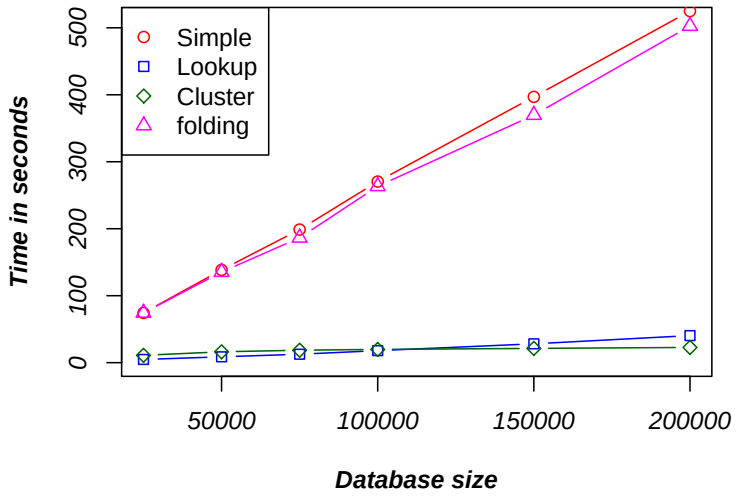

Fig. 4. Experimental results of SPC algorithms on simulated PMF data with varying database size. The vertical axis shows the time in seconds it took to perform 1000 queries.

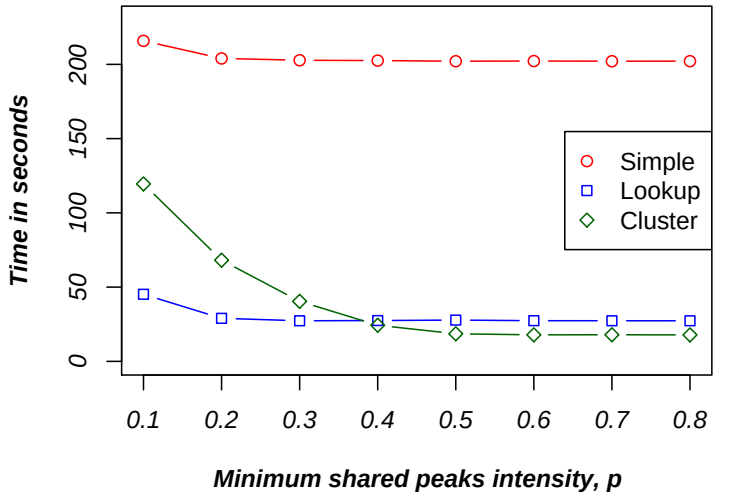

Fig. 5. Experimental results of SPI algorithms on simulated PMF data. The vertical axis shows the time in seconds it took to perform 1000 queries.

ing the overall time of the search. Speeding up database searching is becoming increasingly important due to growing databases and faster data generation.

It is hard to make a good general analysis of the presented algorithms since the running times depend on the distribution of peaks in the spectra in the database and in the query spectrum, and these differ between different databases and different mass spectrometry equipment. So we can not give solid theoretical evidence that our algorithms will always be much faster than the trivial algorithm, but our experiments show that on realistic data our algorithms do give a significant speed-up.

A direction to be explored in the future might be the combination of different of our algorithms. In particular, the folding algorithm and the clustering algorithm are somewhat complementary, such that a hybrid might provide an additional speed-up.

References

1. R. Aebersold and M. Mann. Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207, 2003.
2. D. Dutta and T. Chen. Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. *Bioinformatics*, 23(5):612–618, 2007.
3. J. E. Elias, F. D. Gibbons, O. D. King, F. P. Roth, and S. P. Gygi. Intensity-based protein identification by machine learning from a library of tandem mass spectra. *Nat. Biotechnol.*, 22(2):214–219, 2004.
4. A. Frank, S. Tanner, and P. A. Pevzner. Peptide sequence tags for fast database search in mass-spectrometry. In *Proceedings of RECOMB 2005*, volume 3500 of *Lecture Notes in Bioinformatics*, pages 326–341. Springer Verlag, 2005.
5. M. Havilio, Y. Haddad, and Z. Smilansky. Intensity-based statistical scorer for tandem mass spectrometry. *Anal. Chem.*, 75(3):435–444, 2003.
6. T. Izumi, T. Yokomaru, A. Takahashi, and Y. Kajitani. Computational complexity analysis of set-bin-packing problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E81-A(5):842–849, 1998.
7. J. M. Johnson, J. Castle, P. Garrett-Engele, Z. Kan, P. M. Loerch, C. D. Armour, R. Santos, E. E. Schadt, R. Stoughton, and D. D. Shoemaker. Genome-wide survey of human alternative pre-mRNA splicing with exon junction microarrays. *Science*, 302(5653):2141–2144, 2003.
8. P. J. Kersey, J. Duarte, A. Williams, Y. Karavidopoulou, E. Birney, and R. Apweiler. The international protein index: An integrated database for proteomics experiments. *Proteomics*, 4(7):1985–1988, 2004.
9. M. Mann and O. N. Jensen. Proteomic analysis of post-translational modifications. *Nature Biotechnol.*, 21(3):255–261, 2003.
10. M. Mann and M. Wilm. Error-tolerant identification of peptides in sequence databases by peptide sequence tags. *Anal. Chem.*, 66(24):4390–4399, 1994.
11. M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y. J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, X. V. Gomes, B. C. Godwin, W. He, S. Helgesen, C. H. Ho, G. P. Irzyk, S. C. Jando, M. L. Alenquer, T. P. Jarvie, K. B. Jirage, J. B. Kim, J. R. Knight, J. R. Lanza, J. H. Leamon, S. M. Lefkowitz, M. Lei, J. Li, K. L. Lohman, H. Lu, V. B.

Makhijani, K. E. McDade, M. P. McKenna, E. W. Myers, E. Nickerson, J. R. Nobile, R. Plant, B. P. Puc, M. T. Ronan, G. T. Roth, G. J. Sarkis, J. F. Simons, J. W. Simpson, M. Srinivasan, K. R. Tartaro, A. Tomasz, K. A. Vogt, G. A. Volkmer, S. H. Wang, Y. Wang, M. P. Weiner, P. Yu, R. F. Begley, and J. M. Rothberg. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

12. E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.
13. P. M. Palagi, P. Hernandez, D. Walther, and R. D. Appel. Proteome informatics I: Bioinformatics tools for processing experimental data. *Proteomics*, 6(20):5435–5444, 2006.
14. S. R. Ramakrishnan, R. Mao, A. A. Nakorchevskiy, J. T. Prince, W. S. Willard, W. Xu, E. M. Marcotte, and D. P. Miranker. A fast coarse filtering method for peptide identification by mass spectrometry. *Bioinformatics*, 22(12):1524–1531, 2006.
15. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
16. P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11. IEEE Press, 1973.
17. E. J. Whitfield, M. Pruess, and R. Apweiler. Bioinformatics database infrastructure for biotechnology research. *J. Biotechnol.*, 124(4):629–639, 2006.