Case Study: An Inverted Index for Mass Spectra Similarity Query and Comparison with a Metric-space Method

Rui Mao
Shenzhen University
3688 Nanhai Road, Office Tower #342
Shenzhen, Guangdong, 518060, China
(+86)755 2655 8644
mao@szu.edu.cn

Smriti R. Ramakrishnan
University of Texas at Austin
1 University station, C0500
Austin, TX 78712, USA
(+1)512 471 9541
smriti@cs.utexas.edu

Glen Nuckolls
NetApp
1601 Trapelo Road
Waltham, MA 02451, USA
(+1)512 471 9541
glen.nuckolls@gmail.com

Daniel P. Miranker
University of Texas at Austin
1 University station, C0500
Austin, TX 78712, USA
(+1)512 471 9541
miranker@cs.utexas.edu

ABSTRACT

Query performance is a determining factor in the adoption of an indexing method for similarity query. Metric space indexing methods take great pride in their general applicability. However, it is usually hard for a general method to perform well for every domain. Therefore, it is of interest to investigate the performance of metric-space methods, comparing with domain specific methods, on a particular domain. This paper describes such an investigation for proteomic mass spectra. An inverted index method that exploits the sparsity of mass spectra binary format data and acts as a coarse filter before fine ranking is proposed and empirically compared with an existing metric-space indexing method. Results show that the inverted index method yields greater search efficiency and outperforms the metric-space method in query speed and index size.

Categories and Subject Descriptors

H.2.2 [**Database management**] Physical Design – *Access methods*; H.3.1 [**Information storage and retrieval**]: Content analysis and indexing — *indexing methods*; H.3.3 [**Information storage and retrieval**]: Information search and retrieval — *clustering*, *search process*

General Terms

Algorithms.

Keywords

Similarity query, metric-space indexing, mass spectra, sparse matrix, inverted index.

1. INTRODUCTION

Tandem mass spectrometry (MS/MS) is the mainstream high-throughput technology for detecting proteins in complex samples containing thousands of proteins [10, 14]. A single MS/MS experimental run can generate tens of thousands of spectra, each containing a list of measured mass-to-charge ratios called peaks (real numbers), and the intensity of measurement per peak. Each spectrum represents a fragment of a protein sequence. The task is to identify which proteins are present in the sample, by labeling each spectrum with its correct amino-acid sequence. Multiple identified fragments from a given protein are evidence of its presence in the sample.

We focus on the spectrum identification step of the process, which can be formulated as a similarity search problem in which experimentally generated spectra are matched against a database of idealized "expected" spectra. The mapping is not exact due to several experimental and biological noise factors that necessitate an approximate search framework.

The exponential growth of protein database size [7] necessitates computational methods to analyze spectral data. Most of the analysis time is spent in the database search phase that matches experimental and theoretical spectra. Post-processing to compute peptide and protein scores takes only a few minutes in comparison. As a result, indexing method supporting rapid similarity search of mass spectra has become very popular.

A number of protein identification methods using mass spectra have been developed, such as MASCOT [11], ProFound [20], TruboSEQUEST [18] and clustering method [4]. The similarity measures include cosine distances based on shared peak count [12, 14] and Hausdorff distance [10].

Distance-based indexing [2, 6, 15,19], also known as metric-space indexing, does not make use of data domain information. The available information about the data is derived solely from an oracle that computes the distance between pairs of objects. The only requirement is that the distance function be a metric. The data set can be clustered and a data structure compiled off-line. During an on-line search, the triangle inequality enables the elimination of clusters of data as possible solutions. As a consequence, on-line similarity queries may be quickly computed.

This, so-called, “black-box” model is advantageous for any application where the data cannot be effectively mapped to feature vectors. Even though one is not inclined to use distance-based indexing for multi-dimensional problems, distance-based indexing subsumes multi-dimensional indexing, enabling a uniform programming model for problems that can be recast into metric spaces.

In the context of the MoBIoS project [8], Ramakrishnan et al. propose a metric space method, MSFound, for similarity search of mass spectra [14]. They discuss common similarity measures, and define a cosine-distance based semi-metric distance to compare spectra. They use this distance function in conjunction with MVP Tree [1] to run range and k-NN queries. The system acts as coarse filter to generate a candidate set of results, which may then be ranked by a more discriminative confidence measure in a post-processing step [14]. The top match is regarded as the result.

Dutta and Chen [3] applied another metric-space indexing technique, Locality Sensitive Hashing (LSH), to mass spectra searches. The LSH algorithm [5] is an efficient hash-based indexing structure for nearest neighbor queries that also provides elegant probabilistic bounds on the error of the returned results [17].

The great generality of metric space indexing is also its challenge. It is not hard to imagine that a general method may not perform very well for a particular domain. It is of interest to investigate the performance of metric-space methods on a particular domain, comparing with a method that is specific to that domain. We propose a domain specific indexing method for mass spectra similarity search. This method is compared with MSFound [14] with the same tandem cosine distance.

Mass spectra can be represented in high dimensional binary vector format with sparsity as high as 99.9%. Our method is based on inverted index, a technique commonly used for sparse matrices. Experimental results demonstrate the high search efficiency of our method and its faster search time and smaller index size than MSFound [14].

2. MASS SPECTRA AND TANDEM COSINE DISTANCE

The representation of mass spectra and the tandem cosine distance are already elaborated by Ramakrishnan et al. [14]. They are briefly introduced here for the completeness of this paper.

Each mass spectra data record consists of a single precursor mass (M) and a list of real-valued m/z peaks, P={p_i}.

Given the range of peaks [M_1, M_2] Da, and a resolution $0 \le M_{res} \le$ 1.0Da, the range can be divided into buckets of width M_{res}. Let the buckets be numbered by 0, 1, … Then P can be represented as a high dimensional binary vector S, where S[i] is 1 only if there exists $p \in P$ where p falls into bucket i [b06]. That is:

$$s[i] = \begin{cases} 1, \exists p \in P, i < \dfrac{(p - M_1)}{M_{res}} \le i + 1 \\ 0, otherwise \end{cases}$$

For a common peak range [100, 5000] Da and a resolution 0.2 Da, S is a sparse vector of nearly 25000 dimensions.

This paper deals with mass spectra in its binary format. That is, each mass spectra data record is a pair {M, S}.

Given mass spectra data records A = {M_A, S_A} and B = {M_B, S_B}, the shared peak count SPC(A, B) of A and B is the number of shared 1s of their binary peak vector [14]. That is, SPC(A, B) is the dot product of S_A and S_B:

$$SPC(A, B) = S_A \bullet S_B$$

To account for small peak shifts caused by measurement and calibration error of the mass spectrometer, shared peak count with tolerance, SPC_t, is defined. That is, two shared peaks may not fall into the same bucket, but can be within t buckets [14]:

$$SPC_t(A,B) = \sum_{i:S_A[i]=1} match(i,B)$$

$$match(i,B) = \begin{cases} 1, \exists j \in [i-t, i+t], S_B[j] = 1, \text{ j is not matched with other i} \\ 0, otherwise \end{cases}$$

Note that $SPC_t(A, B)$ is bounded by the number of 1s in S_A or S_B.

Now, the fuzzy cosine distance of A and B, $D_{ms}(A, B)$, is defined as [14]:

$$D_{ms}(A,B) = \arccos\left(\frac{SPC_t(A,B)}{\|S_A\|\|S_B\|}\right).$$

Note that $0 \le D_{ms}(A, B) < \pi/2$.

The precursor mass distance $D_{pm}(A, B)$ computes the difference of precursor mass within a tolerance window [14]:

$$D_{pm}(A,B) = \begin{cases} 0, & |M_A - M_B| \le \tau_{pm} \\ |M_A - M_B|, otherwise \end{cases}$$

A query and its correct match are expected to have common precursor mass and a set of common peak values. In reality, a distance metric must account for measurement errors since experimental peaks can differ from the theoretical database peaks by a few Da depending on the instrument. Therefore, the tandem cosine distance of mass spectra, the distance function advocated by MSFound [14], is defined as a linear combination of fuzzy cosine distance and precursor mass distance [b06]:

$$D_{tcd}(A, B) = C_1 D_{ms}(A, B) + C_2 D_{pm}(A, B).$$

Note that the tandem cosine distance is a semi-pseudometric distance [14].

3. AN INVERTED INDEX METHOD

The key idea of this domain specific method is to take use of the sparseness of the mass spectra binary peak vector. A binary peak vector S can also be represented by a compressed vector $\overline{S}$ of the subscripts of non-zero entries of S in ascending order:

$$\overline{S} = [k_1, k_2, \ldots], S[k_i] = 1.$$

```
$SPC_t(\overline{S_A}[1, 2, \ldots], \overline{S_B}[1, 2, \ldots])$
{
    if ($|\overline{S_A}[1] - \overline{S_B}[1]| \leq t$)
        return $1 + SPC_t(\overline{S_A}[2, \ldots], \overline{S_B}[2, \ldots])$
    else if ($\overline{S_A}[1] < \overline{S_B}[1]$)
        return $SPC_t(\overline{S_A}[2, \ldots], \overline{S_B}[1, 2, \ldots])$
    else
        return $SPC_t(\overline{S_A}[1, 2, \ldots], \overline{S_B}[2, \ldots])$
}
```

Figure 1. Recursive algorithm for SPC_t

Figure 1 gives a short recursive algorithm to compute the shared peak count with tolerance t between two compressed peak vectors $\overline{S_A}$ and $\overline{S_B}$. This algorithm counts the first possible match between two vectors and then eliminates both entries from contributing to a subsequence match. It is not hard to argue the correctness of this algorithm.

Since the above algorithm has to traverse both compressed vectors, one of the performance goals in designing the indexing method is to reduce the number of SPC_t computations.

3.1 Bulkloading the index

Since the computation of precursor mass distance is of low cost and the fuzzy cosine distance is bounded by $[0, \pi/2)$, it is more efficient to compute precursor mass distance first and prune some data before computing fuzzy cosine distance. Therefore, the database is first sorted by precursor mass during bulkload.

An inverted index is maintained to speed up the computation of fuzzy cosine distance. Let N be the dimension of the binary peak vectors and M be the total number of vectors, also let I index the dimensions and j index the vectors. Then an inverted index L is defined as:

$$L = \{L_i \mid L_i = \{j \mid S_j[i] = 1, j = 1, \ldots, M\}, i = 1, \ldots, N\}$$

Compressed vectors
$S_1 = [1, 4]$
$S_2 = [1, 4, 5]$
$S_3 = [2, 4]$
$S_4 = [2]$
$S_5 = [2, 5]$
$S_6 = [5]$
$S_7 = [1, 5]$
$S_8 = [4]$

$\Longrightarrow$

Inverted index
$L_1 = [1, 2, 7]$
$L_2 = [3, 4, 5]$
$L_3 = [\ ]$
$L_4 = [1, 2, 3, 8]$
$L_5 = [2, 5, 6, 7]$

Figure 2. An example of inverted index

That is, for each dimension of the binary peak vectors, a list is stored, consisting of the id of peak vectors whose such dimension is 1. Figure 2 shows an example of inverted index with N = 5 and M = 8. L can be organized as an array of lists to support random access of list heads.

3.2 Range query processing

We consider range query R(q, r) in this section. K nearest neighbor queries can be handled in essentially the same way.

Two pruning rules are exploited to reduce the number of SPC_t computations. They are given as theorems with brief proof in the following.

Theorem 1: (1) A is a query result of range query R(q, r) if $M_q - \max(\tau_{pm}, (r - C_1\pi/2)/C_2) \leq M_A \leq M_q + \max(\tau_{pm}, (r - C_1\pi/2)/C_2)]$ and $r - C_1\pi/2 > 0$; (2) A is not a query result of range query R(q, r) if $M_A > M_q + \max(\tau_{pm}, r/C_2)$, or $M_A < M_q + \max(\tau_{pm}, r/C_2)$.

Proof:

(1) is proved using the fact that $D_{ms} < \pi/2$; (2) is proved using the fact that $0 \leq D_{ms}$. □

1. Prune data using bounds of precursor mass computed from Theorem 1. Put data satisfying Theorem 1 (1) in to a result set, and data satisfying Theorem 1 (2) into a candidate set, together with their precursor mass distance to q.
2. Search inverted index to compute $GSPC_t(q, A)$ for any database point A that $GSPC_t(q, A) > 0$, and A appears in the candidate set.
3. Prune data in the candidate set using Theorem 2.
4. For each element of the candidate set, compute its fuzzy cosine distance using algorithm in Figure 1 to answer the query.

Figure 3. Steps of range query processing

Definition: Given a query mass spectra q, Let L(q) be the set of lists in the inverted index that are within tolerance t to one of q's non-zero entry. That is:

$$L(q) = \{L_i \mid L_i \in L, \exists j, | i - j | \leq t, S_q[j] = 1\}$$

Then the **gross shared peak count with tolerance t** of q and a mass spectra A in the database, $GSPC_t(q, A)$, is defined as the number of appearances of A in lists of (q).

$GSPC_t$ can be easily computed from the inverted index and is used in Theorem 2 for pruning.

Theorem 2: A is not a query result of range query R(q, r) if

$$\arccos\left(\frac{GSPC_t(q, A)}{\|S_q\| \|S_A\|}\right) > \frac{r - C_2 D_{pm}(q, A)}{C_1}$$

Proof:

The fact that $SPC_t(q, A) \leq GSPC_t(q, A)$ is to be used. □

Figure 3 shows the steps in answering a range query R(q, r). Note that in step 1, data is sorted by precursor mass so that fast search methods can be applied. Any 1-dimensional index structure supporting range queries can be applied here. For example, binary search tree for in-memory case and B-tree for large data sets. In step 2, the inverted index is organized as an array of lists, so that the head of each list can be reached in constant time. A hash map can be used to quickly count the gross shared peak count.

Besides query running time, there are some statistics serving as indicators of query efficiency. We show empirical results of them in Section 4.

(1) Precision
At the beginning of step 4, the candidate set contains false positives. Since it is costly to compute the fuzzy cosine distance, we want to minimize the number of positives. The precision is defined as:

$$\text{Precision} = \frac{\text{Number of results found in step 4}}{\text{Candidate set size before step 4}}$$

Larger values of the precision indicate higher query efficiency.

Let pruning efficiency be the percentage of data pruned by a prune rule. Then the pruning efficiency for the two theorems can be defined. Larger values of the pruning efficiency indicate better query performance.

(2) Pruning efficiency of Theorem 1:

$$PE_1 = 1 - \frac{\text{Candidate set size after step 1}}{\text{Database size}}$$

(3) Pruning efficiency of Theorem 2:

$$PE_2 = 1 - \frac{\text{Candidate set size before step 4}}{\text{Candidate set size after step 1}}$$

3.3 Cost analysis

Let N be the dimension of the binary peak vectors, M be the database size, and p be the probability that an entry of a binary peak vector is 1. Random distribution is also assumed.

The expected number of non-zero entries of each binary peak vector, or the expected length of each compressed peak vector, is Np.

The expected size of each list of the inverted index is Mp, and expected total number of ids stored in the inverted index is MNp.

For a given query q and t = 0, the expected number of points with positive gross share peak count to q is given by:

$$M(1-(1-p)^{NP})$$

For tiny values of p, it roughly equals to MNp^2.

(1) Bulkload time

The step to sort the data by precursor mass takes O(MlgM) time. Assuming data input is in compressed vector format, to create the inverted index, one just need to traverse all the non-zero entries. Thus the time is O(MNp). Overall, the bulkload time is O(MlgM).

(2) Space

The space for precursor mass if O(M) and for the inverted index is O(MNp).

Assume an integer takes 4 bytes and a float value takes 4 bytes. The precursor mass array takes 4M bytes. The inverted index takes 4MNp bytes. Therefore, for reasonable data parameters, the index is likely to fit in main memory.

(3) Query running time

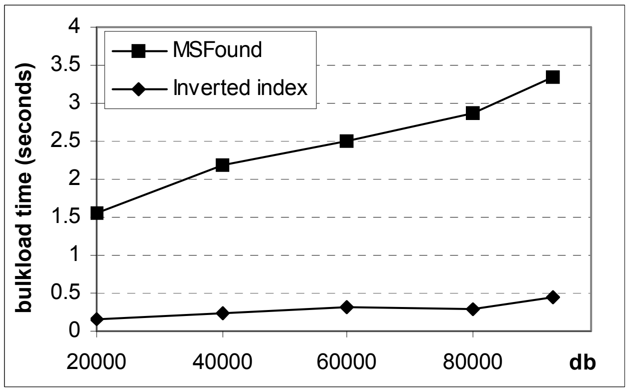
In step 1, determining the range of candidate and result in the sorted data array takes O(lgM) time. Putting candidates into candidate set and computing their precursor mass distances takes O(#candidate) time.

In step 2 for t = 0, the expected number (allow duplicates) of ids in L(q) is Mp*Np = MNp^2. So the total time is O(MNp^2)

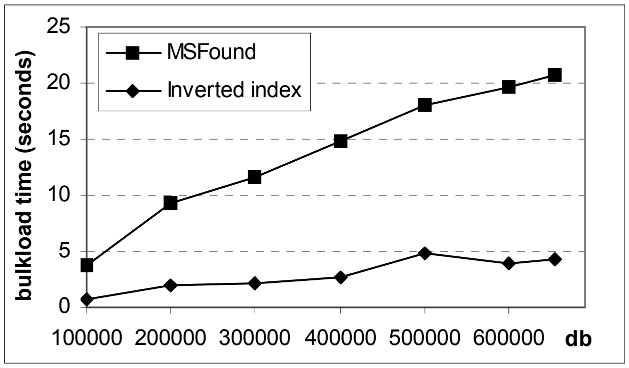
Step 3 takes O(#candidate) time.

In step 4, the shared peak count algorithm (Figure 1) running time is proportional to the length of compressed vectors. Therefore, it is O(NP), and the total time of step 4 is O(NP#computed)

Overall, since #candidate < M, assuming #computed is small (can be empirically verified) and ignoring the I/O cost, the over all cost is O(MNp^2) for t = 0. For sparse vectors, the effect of N and M may be greatly reduced.



(a) Bulkload time of Dataset I: Ecoli



(b) Bulkload time of Dataset II: Human+Ecoli

Figure 4. Bulkload time of two datasets shows that inverted index has up to 10/5X speedups over MSFound

(4) I/O cost

The precursor mass array should be put into continuous disk blocks. The inverted index should also be put into continuous disk blocks while each list takes constant number of blocks to support fast access.

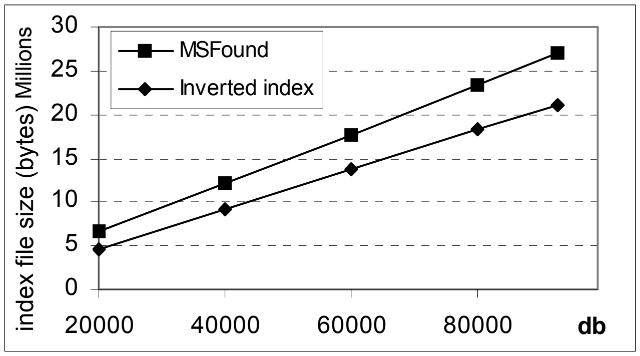
4. EMPIRICAL RESULTS

Two datasets are involved in the empirical study. Dataset I, with size 92768, contains MS/MS spectra from protein sequences of a seven-protein mixture from the Sashimi proteomics repository, concatenated with a control database of spectra from the Escherichia coli K12 (E. coli) genome. Dataset II, with size 654276, combines Database I with a larger control database of theoretical mass spectra from the human genome. A query set of 49 query objects is pre-determined. The test databases and query set in their original format are available from the Open Proteomics Database [13] and the Sashimi mass spectra repository [16].

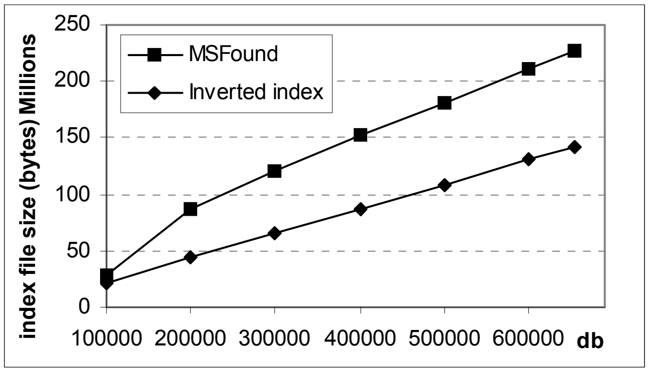
For both datasets, the mass range is [100, 5000] Da and resolution of 0.2 Da is used. So the dimension of the peak vector, N, is about 25000. It turned out that the sparsity of the binary vectors, p, is about 0.1%.

For each dataset, indexes of various sizes, on randomly select data, with both inverted index and general metric space method [14], are built and the bulkload time is collected.

The index file size on disk is also collected as an estimate of space consumption.



(a) Index file sizes of Dataset I: Ecoli



(b) Index file sizes of Dataset II: Human+Ecoli

Figure 5. Index file sizes of two datasets shows that inverted index takes up to 40% less space than MSFound

For each index, range queries of the pre-determined query set with various radii are executed and running time is collected. The running time of brute force linear scan to answer range queries is also collected as the comparison bottom line.

The values of constants are: $C_1 = C_2 = 1$, $\tau_{pm} = 2$, $t = 1$.

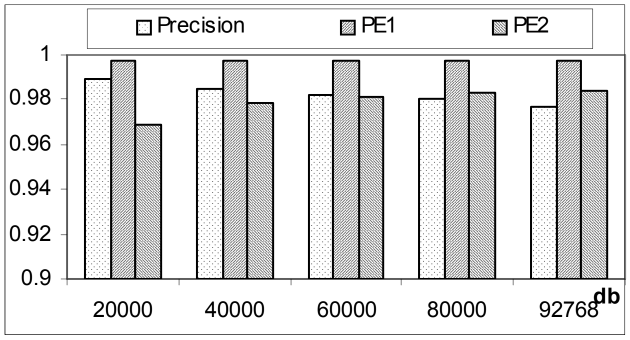
All the algorithms were implemented in JAVA on top of Sun JVM 1.5. All the experiments were carried out on an Apply Mac Pro running Mac OS X Server 10.4 with 8G memory and 2 dual core Intel Xeon CPU at 3 GHZ. The code and data in binary format are available at the MoBIoS repository [9].

4.1 Bulkload time

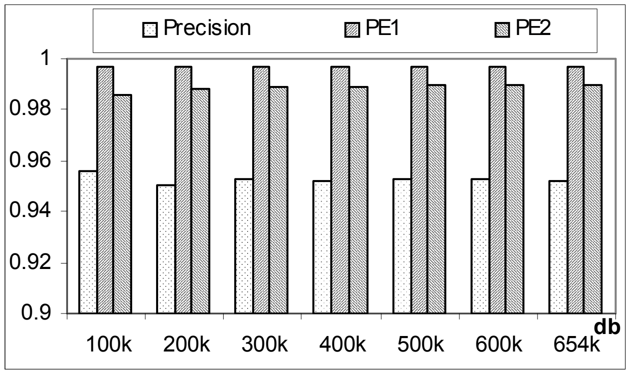
Figure 4 shows the bulkload time of inverted index method and MSFound on two datasets of various sizes. Comparing with MSFound, inverted index has up to 10X speedups on Dataset I and up to 5X speedups on Dataset II. Further, the figure indicates that both methods scales well with respect to bulkload time, consistent with the analysis in Section 3.3.

4.2 Size of index

The space consumption can be estimated by the size of the index file. Sizes of index files of both datasets with various database sizes are illustrated in Figure 5. A linear trend is clearly demonstrated between the index file size and database size, which is consistent with the analysis in Section 3.3. Further, inverted index takes up to 40% less space than MSFound.



(a) Query statistics of Dataset I: Ecoli



(b) Query statistics of Dataset II: Human+Ecoli

Figure 6. Range query statistics show that inverted index has high pruning/searching efficiency

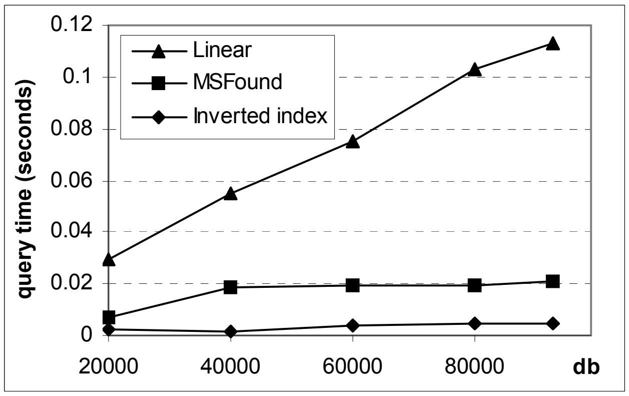
4.3 Range query time

Range queries with various radii are executed on indexes built above, and the query results are averaged on queries. In the following, the results from radius 1.45 are shown, which returns all the biologically correct query results [14].

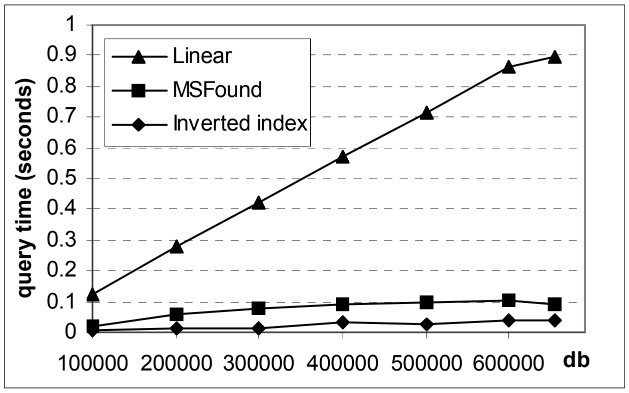
To show the query performance of inverted index, let's first take a look of the empirical results of those query statistics discussed in Section 3.2.

From Figure 6, we can see that the pruning efficiency of Theorem 1, PE_1, is about 99.7% for all the cases. That is, 99.7% of the database is pruned by Theorem 1. The pruning efficiency of Theorem 2, PE_2, is about 98% for Dataset I and about 99% for Dataset II. Therefore, the combined pruning efficiency, $1-(1-PE_1)(1-PE_2)$, of theorems 1 and 2 is as low as 99.994% for Dataset I and 99.997% for Dataset II.

From Figure 6, we can also see the precision is as high as 95% for Dataset I and 98% for Dataset II. That is, a majority of the data not pruned by theorems 1 and 2 is actually true positive. Most of the costly distance computations are performed between the query objects and the actually query results.



(a) Range query running time of Dataset I: Ecoli



(b) Range query running time of Dataset II: Human+Ecoli

Figure 7. Range query running time shows that inverted index outperforms MSFound

Next, let's take a look of the query running time. Figure 7 shows that both inverted index and MSFound are much better than linear scan. Actually, the speedup of inverted index over linear scan is about 20 for both datasets. For MSFound, the speedups over linear scan are about 5 for Dataset I and 9 for Dataset II. Further, the speedups of inverted index over MSFound are about 4 for Dataset I and about 2 for Dataset II, indicating that there are a lot for general metric space methods to catch up with domain specific methods, at least for the case studied here.

Last but not least, the speedups of MSFound over linear scan increases as database size increases for Dataset II. For both datasets, the speedups of inverted index over MSFound decreases as database size increases, indicating better scalability of MSFound.

5. CONCLUSIONS AND FUTURE WORK

The great general applicability of metric space indexing methods does not come for free. It is of interest to investigate the performance of general metric space methods on particular domains, comparing with domain specific method.

Taking use of the high sparsity of binary format mass spectra, we propose a domain specific indexing method, inverted index, for mass spectra to support similarity queries. Empirical results and statistics show that this method has scalable bulkload time and index file size, and great query efficiency.

The inverted index is compared with an existing metric space method, MSFound, and brute force linear scan. Empirical results show that both inverted index and MSFound are much better than linear scan. The inverted index possesses the fastest query time while MSFound has better scalability.

Query performance is one of the most important motivations of the application of metric space indexing method. It is not surprising to see that a domain specific method outperforms a metric space method, as long as the margin is not huge. Our goal is to reveal the cost to be general, and how much metric space methods should improve. Although results show that there is still large space for metric space methods to improve, their better scalability is yet very promising. This paper is just a case study of a particular domain specific method and a metric space method for a particular data type. The test data sets are tiny and we look forward to testing on larger data sets. Further, we expect more work along this direction on more domains, e.g. comparison with Locality Sensitive Hashing [3].

6. ACKNOWLEDGMENTS

This research is supported by NSF DBI-0640923, USA and the Initiating Fund for Member of the Chinese Academy of Sciences from Shenzhen University, China.

7. REFERENCES

[1] Bozkaya, T. and M. Ozsoyoglu, *Indexing large metric spaces for similarity search queries.* ACM Trans. Database Syst., 1999. **24**(3): p. 361-404.

[2] Chavez, E., G. Navarro, R. Baeza-Yates, and J. Marroqu, *Searching in metric spaces.* ACM Computing Surveys (CSUR), 2001. **33**(3): p. 273-321.

[3] 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.

[4] Ari M. Frank, Nuno Bandeira, Zhouxin Shen, Stephen Tanner, Steven P. Briggs, Richard D. Smith, and Pavel A. Pevzner. Clustering Millions of Tandem Mass Spectra. J. Proteome Res. 2008 January; 7(1): 113–122.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In The VLDB Journal, pages 518–529, 1999.

[6] Hjaltason, G.R. and H. Samet, *Index-driven similarity search in metric spaces.* ACM Transactions on Database Systems (TODS), 2003. **28**(4): p. 517-580.

[7] D. Hoksza and T. Skopal. Index-based approach to similarity search in protein and nucleotide databases. CEUR Proc. Dateso 2007, vol. 235, pp. 67-80. 2007.

[8] Miranker, D.P., Xu W. and Mao, R. Mobios: a metric-space dbms to support biological discovery. Proceedings of the International Conference on Scientific and Statistical Database Management System, pp. 241-244, 2003.

[9] The MoBIoS repository: http://aug.csres.utexas.edu/sisap2010_ms

[10] J. Novák, D. Hoksza. Parametrised Hausdorff Distance as a Non-Metric Similarity Model for Tandem Mass Spectrometry. In the Proceedings of the Dateso 2010 Annual International Workshop on DAtabases, TExts, Specifications and Objects. Stedronin-Plazy, Czech Republic, April 21, 2010.

[11] Perkins,D. et al. Probability-based protein identification by searching sequence databases using mass spectrometry data. Electrophoresis, 20, 3551–3567, 1999.

[12] Pevzner,P. et al. Efficiency of database search for identification of mutated and modified proteins via mass spectrometry. Genome Res., 11, 290–299, 2001.

[13] J. Prince, M. Carlson, R. Wang, P. Lu, and E. Marcotte. The need for a public proteomics repository. Nature Biotechnology, 22(4):471–472, 2004.

[14] Ramakrishnan, S. R., Mao, R., Nakorchevskiy, A. A., Prince, J. T., Willard, W. S., Xu, W., Marcotte, E. M., and Miranker, D. P. 2006. A fast coarse filtering method for peptide identification by mass spectrometry. *Bioinformatics* 22, 12 (Jun. 2006), 1524-1531.

[15] Samet, H., *Foundations of Multidimensional and Metric Data Structures*. 2006, Morgan-Kaufmann.

[16] The Sashimi mass spectra repository: http://sashimi.sourceforge.net.

[17] G. Shakhnarovich, T. Darrell, and P. Indyk, editors. Nearest-Neighbor Methods g and Vision: Theory and Practice (Neural Information Processing).The MIT Press, March 2006.

[18] Yates III,J. et al. Method to correlate tandem mass spectral data of modified peptides to amino acid sequences in the protein database. Anal. Chem., 67, 1426–1436, 1995.

[19] P. Zezula, G. Amato, V. Dohnal and M. Batko. Similarity Search: The Metric Space Approach (Advances in Database Systems). Springer, New York, USA. 2006.

[20] Zhang,W. and Chait,B. ProFound—an expert system for protein identification using mass spectrometric peptide mapping information. Anal. Chem., 72, 2482–2489, 2000.