An Efficient Similarity Searching Scheme in Massive Databases

Haiying Shen, Ting Li
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701
{hshen, txl005}@uark.edu

Tom Schweiger
Acxiom Corporation
1 Information Way, Little Rock, AR 72202
Tom.Schweiger@acxiom.com

Abstract

Locality Sensitive Hashing (LSH) is a method of performing probabilistic dimension reduction of high-dimensional data. It is a popular technique for approximate nearest neighbor search. However, LSH needs large memory space and long processing time to achieve good performance when searching a massive dataset. In addition, it is not effective on locating similar data in a very high-dimensional dataset. This paper proposes a new LSH-based similarity searching scheme, namely SMLSH. It intelligently combines a consistent hash function and min-wise independent permutations into LSH. SMLSH effectively classifies information according to the similarity with reduced memory space requirement and in a very efficient manner. It can quickly locate similar data in a massive dataset. Experiment results show that SMLSH is both time and space efficient in comparison with LSH. It yields significant improvements on the effectiveness of similar searching over LSH in a massive dataset.

1. Introduction

Driven by the tremendous growth of information in a massive dataset, there is an increasingly need for an efficient similarity searching method that can locate desired information rapidly with low cost. When querying in a massive dataset, most searching systems generate a high-dimensional vector for each object and then conduct the k-nearest neighbors (KNN) searching [15]. However, such a method is not efficient when the dataset size is very large and the dimension is very high. Other methods [4] [18] [3] [21] [17] [14] [16] relying on a tree structure (such as kd-trees, BDD-trees and vp-trees) require substantial space and time [5] [19]. Sometimes they are even less efficient than the linear search approach [12] which compares a query record with each record in the dataset one at a time. Moreover, all these methods compare a query with records during the searching process to locate similar records, degrading the searching performance.

Locality sensitive hashing (LSH) is a known method that works faster than the linear search for finding nearest neighbors for high-dimensional data [8]. Indyk *et al.* [13] designed a LSH scheme based on p-Stable distributions, which can find the exact near neighbor in $O(\log n)$ time latency, and the data structure is up to 40 times faster than kd-tree [8]. However, the LSH scheme is not effective on locating similar data in a massive dataset with a very high dimension space. In addition, it has low efficiency in terms of memory space and searching speed. An experimental study shows that the LSH scheme requires many hash tables in order to locate most nearest neighbors, and sometimes the LSH may require over a hundred hash tables to achieve reasonable accurate approximations [10]. In addition, the LSH-based method requires all data records have vectors with the same dimension, as it regards records as points in a multi-dimensional space.

We further present a SHA-1 consistent hash function and Min-wise independent permutation based LSH searching scheme (SMLSH) to achieve highly efficient similarity search in a massive dataset. By intelligently integrating SHA-1 and min-wise independent permutations into LSH, SMLSH assigns identifiers to each record and clusters similar records based on the identifiers. Rather than comparing a query with records in a dataset, it facilitates direct and fast mapping between a query and a group of records. The main difference with SMLSH and LSH is that SMLSH does not require all records have the same dimension. LSH needs distance calculation to prune the false positive results which are the records located as similar records but actually are not, while SMLSH does not necessarily to have this refinement step since it incurs much less false positive results. We investigate the operation of LSH and SMLSH, and compare their performance by experiment. Experiment results show that SMLSH enhances LSH's searching efficiency dramatically.

The rest of this paper is structured as follows. Section 2 introduces LSH based similarity searching scheme and min-wise independent permutations. Section 3 describes and an-

alyzes the SMLSH searching scheme. Section 4 shows the performance of SMLSH in comparison with LSH. Section 5 concludes this paper with remarks on possible future work.

2. Related Work

In this section, we introduce LSH, LSH-based similarity searching method [20], and min-wise independent permutations.

2.1. Locality Sensitive Hashing

LSH is an algorithm used for solving the approximate and exact near neighbor search in high dimensional spaces [9] [13] [23]. The main idea of the LSH is to use a special family of hash functions, called LSH functions, to hash points into buckets, such that the probability of collision is much higher for the objects which are close to each other in their high-dimensional space than for those which are far apart. A collision occurs when two points are in the same bucket. Then, query points can get their near neighbors by using the hashed query points to retrieve the elements stored in the same buckets.

LSH provides a dimension reduction technique which projects objects in high-dimensional spaces to lower-dimensional spaces while still preserving the relative distances among objects. Different LSH families can be used for different distance functions. Based on LSH on p-stable

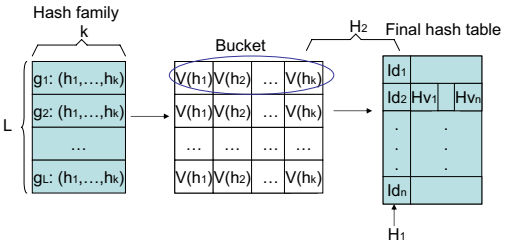

Figure 1. The process of LSH.

distribution [23], a similarity searching method in massive database has been proposed [20]. Figure 1 shows the process of LSH.

First, LSH using Vector Space Model [11] to transform string records into binary numbers. Each binary number denote a dimension.

As shown in Figure 1, LSH then produces the hash buckets $g_i(v)(1 \leq i \leq L)$ by using hash functions $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor$ for every record, where a is a d dimensional vector with entries chosen independently from p-stable distribution and b is a real number chosen uniformly from the range [0,w]. Finally, record v's hashed value by H_2 hash function, H_v, is stored in final hash tables pointed by the hashed value by H_1.

The records that are in the same rows as a query record are the prospecting results of the query. To further refine the results, the Euclidean Space Distance between each located record and the query is computed, and the record will be removed from the located record set if the distance is larger than R, which is a pre-defined threshold of distance.

The following formula is used to compute the Euclidean Space Distance between x and y.

$$d(x, y) = \| x - y \| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

From the procedure of LSH, we can see that LSH does not need to search the query in the entire dataset scope. It shrinks the searching scope to a group of records similar to the query, and conducts refinement. Given n records in a dataset, traditional methods based on tree structures need O(log n) time for a query, and linear searching methods need O(n) time for query. LSH can locate the similar records in O(L) time, where L is a constant. It means that LSH is more efficient in a massive dataset that has rapidly increasing number of records.

2.2. Min-Wise Independent Permutations

Broder *et al.* [6] define that $\mathcal{F} \subseteq S_n$ is min-wise independent if for any set $X \subseteq [n]$ and $x \in X$, when π is chosen at random in $\mathcal{F}$,

$$Pr(min\{\pi(X)\} = \pi(x)) = \frac{1}{\mid X \mid},$$

Where Pr is the probability. All the elements of any fixed set X have an equal chance to become the minimum element of the image of X under π .

In [7] [13], a family of hash functions $\mathcal{F}$ is said to be a LSH function family corresponding to similarity function $sim(A, B)$ if for all $h \in \mathcal{F}$ operating on two sets A and B, we have:

$$Pr_{h \in \mathcal{F}}[h(A) = h(B)] = sim(A, B),$$

where $sim(A, B) \in [0, 1]$ is a similarity function.

3. SMLSH Searching Scheme

A massive dataset has tremendous number of keywords, and a record may contains only a few keywords. As a result, in LSH, the identifier of a record may have a lot of 0s, and only a few 1s. This identifier sparsity leads to low effectiveness of Euclidean Space Distance measurement to quantify the closeness of two records. This is confirmed by our simulations results that the LSH returns many records that are not similar to the query even though all expected records are returned. We also observe that the memory required for LSH algorithm is mainly used to store the identifiers of records and the hash tables.

SMLSH reduces the false positive results and meanwhile reduces the memory for records and hash tables. It does not require all records have the same dimension. That is, it does not need to derive a vector for each record from a unified multi-dimensional space consisting of keywords. Specifically, it uses SHA-1 consistent hash function to generate an identifier for each keyword in a record. SHA-1 hash function is supposed to be collision-resistant [22], so it can be used to hash keywords into integers. Since SHA-1 distinguishes uppercase and lowercase keywords. SMLSH firstly changes all keywords to uppercase. As shown in the following, after changing all the keywords of a record into capital letters, SMLSH uses SHA-1 to hash all the capital-letter keywords to a set of integers:

Original record: ANN | EDNA | Shelby | NC | 0541

Uppercase record: ANN | EDNA | SHELBY | NC | 0541

Hashed record: 1945773335 | 628111516 | 2140641940 | 2015065058 | 125729831

Algorithm 1. Pseudo-code for the procedure of SMLSH.

(1) get n groups of m hashed values of a and b
(2) **for** each k[i] **do** *//k[i] is one of the keywords of a record*
(3) using SHA-1 hash k[i] into hashK[i]
(4) **for** each a[p][q] and b[p][q] **do**
(5) $g[p][q] = (a[p][q] * hashK[i] + b[p][q]) \bmod prime$
(6) **if** i == 0 **then**
(7) min[p][q]=g[p][q]
(8) **else if** $g[p][q] < min[p][q]$ **then**
(9) min[p][q] = g[p][q]
(10) **endif**
(11) **endif**
(12) **endfor**
(13) **endfor**
(14) **for** each hashID[j] **do**
(15) hashID[j]=0
(16) **for** each min[j][t] **do**
(17) hashID[j]^=min[j][t]
(18) **endfor**
(19) hashID[j]=hashID[j]mod tableSize
(20) inseart the index of the record into the hash table
(21) **endfor**
end

SMLSH does not like LSH require that all records have the same dimension. In SMLSH, the length of a record vector only equals to the number of keywords in itself. Thus, SMLSH reduces the memory of LSH for vectors. In SMLSH, the min-wise independent permutations are defined as: $\pi(x) = (ax + b) \bmod prime$,
where a and b are random integers, $0 < a \leq prime$ and $0 \leq b \leq prime$. SMLSH makes n groups of m min-wise independent permutations. Applying the $n \times m$ hash values to a record, we will get n buckets with each bucket having m hashed values. SMLSH then uses XOR operation on the values of each bucket to get a final hash value. Consequently, each record will have n final hashed values, denoted by *hashID*.

$hashID_i = (min\{\pi_1(S')\}\, XOR\, min\{\pi_m(S')\}) \bmod tableSize,$

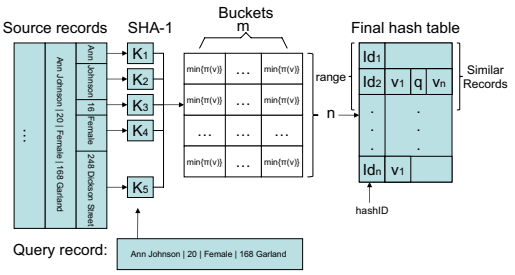

Figure 2. Searching process of SMLSH.

where S' is a SHA-1 hashed integer set, $1 \leq i \leq n$. Algorithm 1 shows the pseudo-code for the procedure of SMLSH.

For a record searching, SMLSH gets the hashIDs for the query record based on the algorithm. It then searches the hash table, exports all the records with the same hashID as the similar records of the query record. A range also can be set to enlarge the searching scope. With the range, the records with hashID[j] that satisfy condition:

$$\mid hashID[j] - RhashID \mid \leq range$$

are also checked, where $RhashID$ is the hashID of a query record. This method enlarges the searching scope of similar records, and increase the searching accuracy. Figure 2 shows the searching process of SMLSH.

To enhance the accuracy of returned similar records, refining can be conducted based on similarity. That is, the similarity between each returned record and the query record is calculated, the records whose similarities are less than a pre-defined threshold are removed from the list. Given two records A and B, the similarity of B to A is calculated as follows:

$$similarity = \frac{\mid A \cap B \mid}{\mid A \mid}$$

For example,

A: Ann | Johnson | 16 | Female

B: Ann | Johnson | 20 | Female

To A, the similarity of B is $\frac{3}{4} = 0.75$.

4. Performance Evaluation

We implemented the SMLSH searching system, and compare it with E2LSH 0.1 of MIT [1]. E2LSH 0.1 is a simulator for the high-dimensional near neighbor search based on LSH in the Euclidean space. Our testing dataset is a set of synthetically generated names and addresses obtained from Acxiom Corporation that imitates the properties of actual customer data. After each record is transformed to $0/1$ binary dataset, the dimension of the record is 20,591. The number of source records was 10,000. We selected 97 query records randomly from the source records. We use *target records* to denote the records in the dataset that are similar

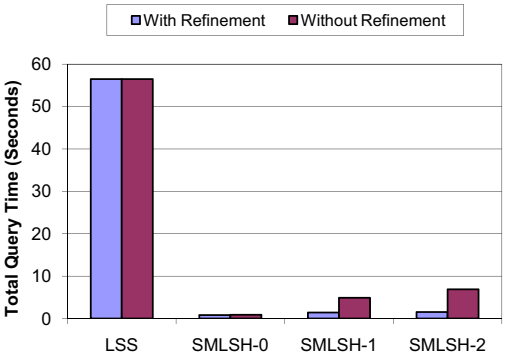

Figure 3. Total query time.

to the query record. In the hash function $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor$ of LSH, w was set to 4 as an optimized value [2]. The distance threshold of R was set to 3 in all experiments. In SMLSH, we changed different m and n to test the result. At last, m and n were set to 4 and 20.

We compared the performance of SMLSH with LSH in terms of query time, memory cost and effectiveness in locating the similar records. We conducted experiments for the following methods:

(1) LSH, denoted as LSS;

(2) SMLSH with $range = 0$, denoted as SMLSH-0;

(3) SMLSH with $range = 8$, denoted as SMLSH-1;

(4) SMLSH with $range = 16$, denoted as SMLSH-2;

Unless otherwise specified, all these methods don't have refinement phase.

4.1. Query Time

Figure 3 shows the total query time of different methods with and without refinement. We can see that SMLSH has much faster query speed than LSH. This is due to two reasons. First, LSH needs to conduct more hash value calculations than SMLSH. In LSH, there are 2346 groups of buckets, and 69 hash functions in each group. In SMLSH, there are 20 groups of buckets, and 4 hash functions in each group. Therefore, LSH needs to do much more hash value calculations than SMLSH. Second, LSH does Euclidean Space Distance computation, which includes multiple operations: addition, subtraction and square calculation to remove its false positive results. SMLSH-0, SMLSH-1 and SMLSH-2 do not need this refinement phase since they generate much less false positive results.

In Figure 3 We set the similarity threshold for SMLSH as 0.5. That is, SMLSH will return the records whose similarity to the query record are no less than 0.5. From Figure 3 we can observe that the query time with refinement is more than that without refinement. It is because the refinement phase needs time for similarity calculation. With refinement phase, SMLSH still is much faster than LSH in searching. The query time of SMLSH-2 is longer than SMLSH-1, and

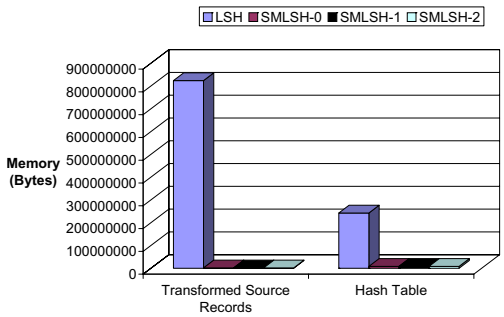

Figure 4. Memory cost for source record and hash tables.

that of SMLSH-1 is longer than SMLSH-0. It is expected since larger range means more hash values needed to be checked, and more similarity calculations need to be conducted in the refinement phase.

4.2. Memory Cost

Recall that LSH transforms source records to vectors based on keyword list, and SMLSH uses SHA-1 to get record vectors. Both of them need memory space for record vectors and hash tables. Figure 4 shows the memory size for storing transformed source records and hash tables of LSH and different SMLSHs. It demonstrates that the memory consumption for both transformed source records and hash tables in SMLSH is much smaller than in LSH. It is due to the reason that SMLSH has much shorter record vectors and hence less storage memory. The vector dimension of LSH is 20,591, while the average dimension of SMLSH is 11. Therefore, SMLSH needs less memory for storing the transformed source records than LSH. There are 2346 groups of buckets in LSH for each record, so there are 2346 hashed values needed to be saved in the hash table for each record. For 10,000 source records, the hash table should save 23,460,000 hashed values totally. SMLSH only has 20 groups of buckets for each record, and the total number of hashed values contained in hash table is 200,000. Consequently, LSH's hash table size is about 117 times more than SMLSH's hash table size. These results verify that SMLSH can significantly reduce the memory consumption of LSH.

4.3. Effectiveness

In addition to the efficiency in terms of memory consumption and query time. Another important metric for searching methods is how many target records are missed in the returned record set. This metric represents the effectiveness of a searching method to locate target results. We define *accuracy* as follows:

$$Accuracy = \frac{Total\ number\ of\ target\ records\ located}{Total\ number\ of\ target\ records}$$

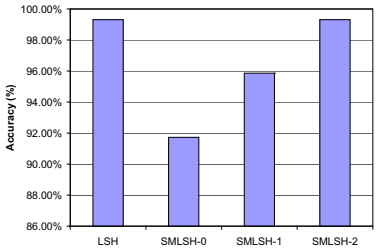

Figure 5. Accuracy.

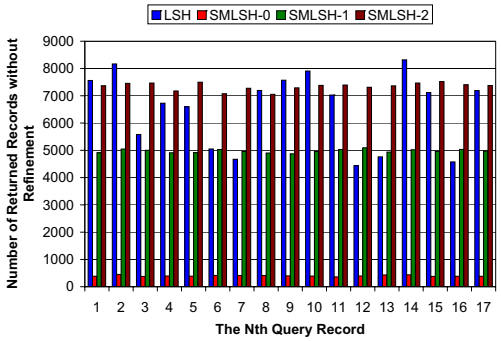

Figure 6. The number of returned results without refinement.

and tested the performance of searching methods in terms of accuracy.

Figure 5 shows the accuracy for each method. We observe that LSH and SMLSH-2 have higher accuracy than others, and they can find nearly all of the target records. However, the accuracy of SMLSH-0 and SMLSH-1 is lower than LSH and SMLSH-2. Since SMLSH-0 and SMLSH-1 have smaller range scope of the query record to check the similar records, they may miss some similar records that have less similarities to the query record. Therefore, with an appropriate value of range, SMLSH can achieve comparable performance as LSH, but at a dramatically higher efficiency.

Figure 6 plots the number of returned results without refinement. An effective method should return fewer false positive records. LSH and SMLSH-2 returned about 2/3 of source records before refinement phase. They return much more records than SMLSH-0 and SMLSH-1. More returned records reduces the possibility to miss target prospects. This is the reason why LSH and SMLSH-2 have high accuracy. SMLSH-0 and SMLSH-1 has less range, they return less records but with lower accuracy. The results imply that an appropriate range is very important to the performance of SMLSH. The range should be set to a value that will lead to few target prospect misses, and meanwhile will not incur many returned records.

In order to see the similarity degree of located records to the query record of SMLSH, we conducted experiments on SMLSH-0, SMLSH-1 and SMLSH-2. We randomly chose one record, and changed one keyword to make a new record

Table 1. Whether the original record can be found.

Similarity	SMLSH-0	SMLSH-1	SMLSH-2
1.0	Y	Y	Y
0.9	Y	Y	Y
0.8	Y	Y	Y
0.7	Y	Y	Y
0.6	N	N	Y
0.5	N	N	Y
0.4	N	N	Y
0.3	N	N	Y
0.2	N	N	Y
0.1	N	Y	Y

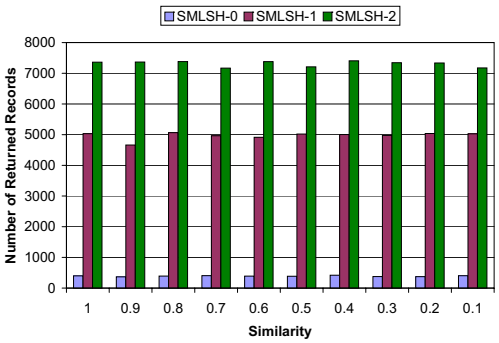

Figure 7. The number of returned record versus similarity.

as query record every time. Our purpose is to see if SMLSH can find the original record with the decreasing degree of similarity to the query record. Table 1 shows whether the method can find the original record when it has different similarities to the query record. "Y" means the method can find the original record and "N" means it cannot. The figure illustrates that SMLSH-2 can locate the original record in all similarity levels, and SMLSH-0 and SMLSH-1 can return the records whose similarity are greater than 0.6 to the query record. The reason that SMLSH-2 can locate records with small similarity is because it has larger scope of records to check. The results imply that in SMLSH, records having higher similarity to the query record have higher probability to be located than records having lower similarity.

Figure 7 depicts the number of returned records versus the similarity between the query record and the original record. From the figure, we can see that the change of similarity does not affect the number of returned records. Every time, each method returns almost the same number of records. Due to the larger searching scope, SMLSH-2 returns more records than SMLSH-1, and SMLSH-1 returns more records than SMLSH-0. These results further confirm the importance to choose an appropriate value of range, to decrease false positives and meanwhile avoid missing similar records.

5. Conclusions

Traditional information searching methods rely on linear searching or a tree structure. They need to compare a query with the records in the dataset in the searching process, leading to low efficiency. This paper first describes a Locality Sensitive Hashing (LSH) based similarity searching, which is more efficient than linear searching or tree structure based searching in a massive dataset. However, it still needs a large memory space for storing source record vectors and hash tables, and leads to long searching latency. In addition, it is not effective in a very high-dimensional dataset. This paper further presents an improved LSH based searching scheme (SMLSH) that can efficiently and successfully conduct similarity searching in a massive dataset. SMLSH integrates SHA-1 consistent hashing function and min-wise independent permutations into LSH. It avoids sequential comparison by clustering similar records and mapping a query to a group directly. Moreover, compared to LSH, it cuts down the space requirement for storing source record vectors and hash tables, and accelerates the query process dramatically. Simulation results demonstrate the efficiency and effectiveness of SMLSH in similarity searching in comparison with LSH. SMLSH dramatically improves the efficiency over LSH in terms of memory consumption and searching time. In addition, it can successfully locate queried records. Our future work will be focused on further improving the accuracy of SMLSH.

Acknowledgments

This research was supported in part by the Acxiom Corporation.

References

[1] A. Andoni. Lsh algorithm and implementation (e2lsh), 2005. http://web.mit.edu/andoni/www/LSH/index.html.

[2] A. Andoni and P. Indyk. E2lsh 0.1 user manual, 2005. http://web.mit.edu/andoni/www/LSH/index.html.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, Arlington, Virginia, USA, January 23-25,1994.

[4] J. L. Bentle, J. H. Friedman, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[5] C. Bohm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, (3):630–659, 2002.

[7] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. of the 34th ACM Symposium on Theory of Computing*, pages 380–388, Montral, Qubec, Canada, May 19-21, 2002.

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of the 20th annual symposium on Computational geometry (SCG)*, New York, NY, USA, June, 2004.

[9] M. Datar, N. Immorlica, P. Indyk, and V.Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of DIMACS Workshop on Streaming Data Analysis and Mining*, New Jersey, USA, March 24-26, 2003.

[10] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. *The VLDB Journal*, pages 518–529, 1999.

[11] D. A. Grossman and O. Frieder. *iFlow: Information Retrieval*. Springer, The Netherlands, USA, 2004.

[12] J. J. Hu, C. J. Tang, J. Peng, C. Li, C. A. Yuan, and A. L. Chen. A clustering algorithm based absorbing nearest neighbors. In *6th International Conference of WAIM*, Hangzhou, China, October 11-13, 2005.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of the 13th annual ACM Symposium on Theory of Computing*, pages 604–613, Dallas, Texas, USA, May 23-26, 1998.

[14] J. B. Kruskal and M. Wish. *Multidimensional scaling*. SAGE publication, Beverly Hills, 1978.

[15] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Integrating semantics-based access mechanisums with p2p file systems. In *Proc. of the the Third International Conference on Peer-to-Peer Computing (P2P)*, Linkping, Sweden, September 1-3,2003.

[16] M. L. Miller, M. A. Rodriguez, and I. J. Cox. Audio fingerprinting: Nearest neighbor search in high dimensional binary spaces. In *IEEE Multimedia Signal Processing Workshop*, St. Thomas, US Virgin Islands, December 9-11, 2002.

[17] C. W. Niblack, R. Barber, W. Equitz, M. D. Flickner, E. H. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: querying images by content using color, texture and shape. In *Proc. of SPIE: Storage and Retrieval for Image and Video Database*, April, 1993.

[18] R. Panigrahy. Nearest neighbor search using kd-trees. Technical report, Stanford University, 2006.

[19] T. Sellis, N. Roussopoulos, and C. Faloutsos. Multidimensional access methods: Trees have grown everywhere. In *Proc. of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 25-29, 1997.

[20] H. Shen, T. Li, Z. Li, and F. Ching. Locality sensitive hashing based searching scheme for a massive database. In *IEEE SoutheastCon 2008*, Huntsville, AL, USA, April 3-6, 2008.

[21] D. A. White and R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, University of California, 1996.

[22] Y. Zhu, H. Wang, and Y. Hu. Integrating semantics-based access mechanisums with p2p file systems. In *Proc. of the the Third International Conference on Peer-to-Peer Computing (P2P)*, Linkping, Sweden, September 1-3, 2003.

[23] V. M. Zolotarev. *One-Dimensional Stable Distributions*. American Mathematical Society, 1986.