Token List Based Data Searching in a Multi-Dimensional Massive Database

Ting Li, Haiying Shen, Anthony M. Rosequist
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701

Abstract – *A massive database contains a large volume of data. Therefore, how to efficiently search data in a massive database has been a challenge. This paper proposes token list based data searching scheme (TLS) which abstracts tokens from the database, and clusters the data based on their tokens. Query only needs to be conducted on a small group of data records, rather than all the records in a database. Experiment is conducted on a database by using TLS. The experiment results show that TLS is an efficient database searching scheme in terms of memory consumption, search latency and accuracy in locating data.*

Keywords: Data Search, Proximity Search, Locality Sensitive Hashing

1 Introduction

The rapid growth of information nowadays leads to the increase of the size of a database, which has posed a challenge for scalable information retrieval for desired data. Imagine a person is in a large library that holds millions of books. If he does not know the book number or the name of the book, he probably needs to search the entire library for the desired book. However, if the books have been classified into different categories, he only needs to work in the certain category to find the desired book. Book classification saves him searching time.

Proximity information search is to search information in a database similar to a query. Many websites supply proximity information search. For example, Google can not only search the exact information on the web according to the keywords supplied by a user, but also find proximity information which contains the specified keywords. Therefore, an efficient classification method can reduce the searching time in a massive database and locate most of the data related to the query. One approach for data searching is to cluster similar data. There are many clustering methods such as K-means clustering [1] and fuzzy c-means clustering [2] [3]. Some of these clustering methods are built on tree-structure and they rely on distance measurement like Euclidean Space Distance measurement, which contains complex multiple operations: minus, square and root sum square, these operations increase the query latency. In addition, the maintenance of tree-structure brings about a cost of high overhead.

This paper presents a token list based data searching scheme (TLS) that avoids using tree-structure and distance measurement. By classifying records base on token list, TLS shortens query time, reduces memory consumption and also works for different dimensional data sets. TLS saves all the unique tokens contained in the records into a token list. Then it groups records according to their tokens. When query a record, TLS directly map the query to the groups which have the query record's tokens.

This paper analyzes the potential of TLS for information searching in a database and details the design of TLS. The paper also discusses the process for data insertion and deletion in a database. We applied TLS on a database to evaluate its performance in comparison with other information searching schemes. The experimental study shows that TLS is effective in categorizing data and searching proximity information in a massive database. TLS also achieves high accuracy on searching proximity information, and requires much less memory and query time.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative methods for information searching. Section 3 describes the design of TLS. Section 4 evaluates the performance of TLS in comparison with other searching schemes. Section 5 concludes this paper.

2 Related work

There are many proximity information searching methods. The simplest one is linear search. It compares a query with all the source records in a database one at a time. Therefore, linear search leads to long query time. Another method for proximity information searching relies on tree-structure [4, 5, 6, 7, 8], such as kd-trees, BDD-trees and vp-trees. Tree-structure proximity information searching method reduces the searching scope of linear search method since only partial records need to be compared with query record. However, tree-structure incurs high maintenance cost.

Clustering methods have been proposed to accelerate the relevant information searching speed in a database. Generally, there are two categories of clustering methods [18]:

hierarchical clustering [9] and partitional clustering methods [1, 2, 10, 11].

Hierarchical clustering algorithm [9] establishes a dendrogram tree in which each individual element, i.e. record, is a leaf. The elements are grouped into a cluster according to a distance metric. The new parent cluster consists of close elements in distance. This process returns a combination of clusters, also known as a tree, that contain all the elements as the root. Cutting a tree at different heights will return different clustering combinations [18].

Unlike hierarchical clustering algorithm, partitional clustering algorithms determine all the clusters at once. Examples of partitional clustering include K-means clustering [1], fuzzy c-means clustering [2], QT clustering [10] and locality-sensitive hashing [11].

K-mean clustering [1] method assigns each point to the cluster whose center is the nearest to the point. First, it decides the number of clusters; second, it randomly generates the specified number of clusters, and gets the centers of clusters; third, it assigns each point to the nearest cluster center and recomputes the cluster center. It repeats the processes until the assignment doesn't change [1, 18]. The advantage of K-mean clustering method is dataset fast clustering. Its disadvantage is that it cannot ensure that the result has a global minimum of variance [9, 18].

Fuzzy c-means clustering [2] is very similar to k-mean clustering algorithm. In fuzzy clustering, a point has a degree of belonging to a cluster. According to this degree, each point is assigned to a number of clusters. The degree is calculated after the assignment. When the difference between degrees of two iterations is no more than a given threshold, the clustering completes. Fuzzy c-means clustering has the same problem as K-mean clustering algorithm [18].

Heyer et. al. [10] proposed quality threshold (QT) clustering which does not require specifying the number of clusters as a priori. It is used for gene clustering. QT clustering method lets the user choose a threshold diameter for clusters, and then builds a candidate cluster for each point by including the closest points, until the diameter threshold is met. The largest candidate cluster, with the user-specified minimal number of points, is selected as a QT cluster. After a QT cluster is generated, QT clustering method removes all the points in the QT cluster from the consideration of the next QT cluster's generation, and repeats the process with the remaining points. As a result, a set of QT clusters are generated, and the points that do not belong to any QT clusters are grouped together as "unclassified" group [10, 19]. LSH [12, 13, 14, 15, 16, 17] can also be used for clustering; therefore, using a set of hash functions to group similar points into groups.

3 Token list based searching scheme

Token list based searching scheme (TLS) is designed for finding proximity information in a massive database. TLS first builds a token list to collect all the unique tokens in the records. Then, TLS maps each record to the token list according to its component tokens, and records the record's index at each token in the token list. A record's index specifies the location of the record in a database. In searching, TLS maps a query to the token list based on the query record's tokens, and retrieves all indices of records having one of the query's tokens. By classifying records base on token list, TLS can shorten query time, reduce memory consumption and also work for different dimensional data sets.

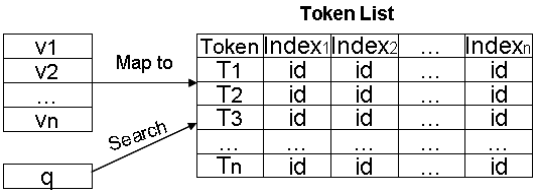

Figure 1: An example of TLS process.

3.1 TLS description

Figure 1 shows the process of TLS. The *id* in the figure presents the index of a record. Each row in the token list consists of a unique token in the database and indices of records having the token. To build the token list, TLS reads all the records in the database, and derives the unique tokens into the token list. At the same time, TLS saves the indices of records into the token list according to their component tokens. To search a query, TLS directly goes to the row of each of the query's tokens in the token list. The records whose indices are in the rows of the query record's tokens are returned as the similar records of the query records.

An example is used to explain the TLS information searching process. Assume that the records in a database are as follows: the index of record id_1 is 1; the index of record id_2 is 2; the index of record id_3 is 3; the index of record id_4 is 4.

id_1: Ann Johnson | 16 | Female | 248 Dickson Street
id_2: Ann Johnson | 20 | Female | 168 Garland
id_3: Mike Smith | 16 | Male | 1301 Hwy
id_4: John White | 24 | Male | Fayetteville | 72701

Using the token list construction method, TLS reads the records id_1, id_2, id_3 and id_4, and builds a token list table as shown in Figure 2. As a result, all records are grouped by their tokens. That is, the indices of records having a common token are grouped in one cluster. For example, the indices of the records id_1 and id_3 are clustered based on the token "16". Therefore, those records are considered as similar records to a certain degree.

Token	Index	Index
16	1	3
20	2	
24	4	
168	2	
248	1	
1301	3	
72701	4	
ANN	1	2
DICKSON	1	
FAYETTEVILLE	4	
FEMALE	1	2
GARLAND	2	
HWY	3	
JOHN	4	
JOHNSON	1	2
MALE	3	4
MIKE	3	
SMITH	3	
STREET	1	
WHITE	4	

Figure 2: An example of building token list table.

Next, we will introduce how TLS processes a query. Assume a query record is:

q: Ann Johnson | 20 | Female | 168 Garland

Because the query record *q* has token "2", "168", "ANN", "FEMALE", "GARLAND" and "JOHNSON", TLS checks the collections under "2", "168", "ANN", "FEMALE", "GARLAND" and "JOHNSON" tokens. Figure 3 shows the $collection_1(C_1)$, C_2, C_3, C_4, C_5 and C_6 are the groups of records which have one token in record *q*. Then, TLS unions these collections:

$$SimilarRecord = C_1 \cup C_2 \cup C_3 \cup C_4 \cup C_5 \cup C_6 \quad (1)$$

	Token	Index	Index
	16	1	3
1	20	2	
	24	4	
2	168	2	
	248	1	
	1301	3	
	72701	4	
3	ANN	1	2
	DICKSON	1	
	FAYETTEVILLE	4	
4	FEMALE	1	2
5	GARLAND	2	
	HWY	3	
	JOHN	4	
6	JOHNSON	1	2
	MALE	3	4
	MIKE	3	
	SMITH	3	
	STREET	1	
	WHITE	4	

Figure 3: An example of searching token list.

The records with id_1 and id_2 are returned since their indices are linked with the tokens of *q* in the token list. Therefore, records with id_1 and id_2 are similar records to *q*.

3.2 Data inserting and deleting operations

In addition to data searching, inserting and deleting are two important operations in a database. Insertion operation is to store a new record into a database; Deletion operation is to remove a record from a database.

When inserting a new record into a database, the new record's tokens will be scanned. If a token of the new record has been stored in the token list, the index of the new record will be linked with the token in the token list. Otherwise, the new token will be added into the token list. At the same time, the index of the new record will be also linked with the new token.

When a record needs to be deleted from a database, TLS scans the record and locates the tokens of the record in the token list. TLS then deletes the index of the record linked with the tokens. If the index of the record is the only one linked with a token, the token is also deleted from the token list to release the space taken by the record. As a result, there are no indices of the record in the token list. The record is completely deleted from the database.

4 Performance evaluation

We implemented the proposed TLS system, and conducted the experiments on TLS. Our testing data set is a set of synthetically generated names and addresses that imitates the properties of actual customer data in a database. There are 20,591 unique tokens in the database. There are totally 10,000 source records in the database. We randomly selected 97 query records from source records. Each record has a different number of tokens, the average number of the tokens in a record is 10.

4.1 Memory consumption

In this experiment, we compared TLS with LSH in terms of memory consumption. We used E2LSH 0.1 [14] for LSH simulation. E2LSH 0.1 is a simulator for the high dimensional near neighbor search based on LSH in the Euclidean space developed by MIT.

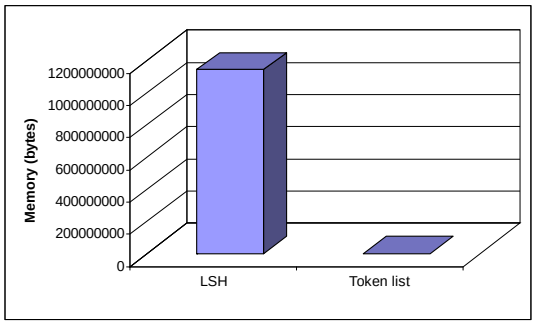

Figure 4: Total memory consumption of LSH versus TLS.

Figure 4 plots total memory requirement of LSH and TLS. From Figure 4, we can see that LSH incurs much higher memory consumption than TLS. LSH uses Vector Space Model (VSM) [13] to transform each record to a 0/1 binary number identifier. Because the number of unique tokens is 20,591, the length of the binary number string is 20,591 for every record. LSH can only process record identifiers rather than strings. Moreover, it requires that all the binary number identifiers must have the same dimension (i.e. 20,591). In contrast, TLS does not have the requirement of the same

numbers of tokens. The average number of the tokens of a record is 10. Therefore, in TLS, the number of index saved in the token list for a record is 10 on average. This leads to much less memory for storing the source records. In addition, LSH generates a number of hash tables, while TLS only has one hashed token list table. Therefore, LSH's memory consumption is significantly higher than TLS's.

4.2 Time consumption

Figure 5 shows the query times of linear searching, kd-tree searching, and TLS versus the number of query records, respectively. We can see that the query time of the linear searching method is the highest, and TLS leads to faster similar record searching than kd-tree method. Given n pieces of records in a database, traditional methods based on tree structures need $O(\log n)$ time for a query, and linear searching methods need $O(n)$ time for a query. TLS need $O(T)$ time to locate all the similar records, where T is a constant related to the number of tokens contained in a query record. Because the value T is much less than the value of n, the total query time of TLS is much less than linear search and kd-tree searching.

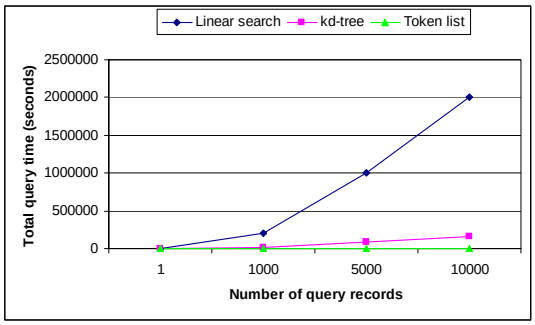

Figure 5: Total query latency of linear searching, kd-tree searching, and TLS.

4.3 Effectiveness

False positive results are those records that are located as similar records but actually are not. An effective method should return fewer false positive records.

Figure 6 presents the number of records returned by a query. In the figure, the data for "actual similar records" is gotten from linear search. We compared each query record with every record in the database to decide if the source record is the similar record of query record. From Figure 6 we can observe that TLS can cluster the similar records according to their tokens and it also can find all the similar records of the query record. There is no false positive in the returned records of TLS. Because TLS uses the tokens of query records to query the corresponding position of the tokens in the hashed token list table, it ensures that the returned records are all true similar records of the query that have at least one token the same as a token in the query.

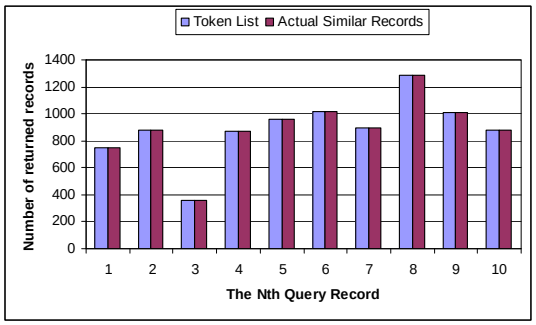

Figure 6: Number of returned record.

4.4 Accuracy

An important metric for a searching scheme is accuracy, which is used to measure how many actual similar records a scheme will miss in searching. A scheme with higher accuracy misses less similar records. Figure 6 shows that TLS can return 100% of similar records for each query. Because the query is conducted by searching the tokens of the query record in the token list, only the records having the same token as the query record can be returned. Therefore, TLS has high accuracy.

Another accuracy experiment is conducted on TLS. We randomly chose one record from the 10,000 records, and changed one token of the selected records every time to make a new record as the query record. Therefore, every generated record will have different similarity from 1.0 to 0.1. The new generated query records are inputted into TLS to search the similar records in the 10,000 source records to test if they can find the original selected record. The result is shown in Table 1. If the original selected record can be found, "Y" is marked, otherwise, "N" is marked.

From Table 1 we can see that TLS can find all the original selected record even their similarity as low as 0.1. The reason is that TLS groups records according to their token.

Table1: Data searching accuracy.

Similarity	TLS
1.0	Y
0.9	Y
0.8	Y
0.7	Y
0.6	Y
0.5	Y
0.4	Y
0.3	Y
0.2	Y
0.1	Y

From the experiments, we can get the conclusions:

(1) TLS improves the search speed of linear search and tree structure search.

(2) TLS outperforms LSH in terms of memory consumption. Compared to LSH, TLS consumes much lower memory.

(3) TLS has high accuracy; it can locate all the similar records of a query.

(4) TLS does not have requirement on the dimension of the datasets. All the records in the database can have different number of tokens.

5 Conclusions

In order to reduce the overhead and query latency of proximity information searching in a database, this paper presents a token list based proximity data searching scheme (TLS) that can successfully and efficiently search proximity information in a massive database. TLS builds a token list contains all the unique tokens in the records. It groups the records having the same token together. A query is only mapped to the groups linked to the query's tokens. Experiment results show the high performance of TLS compared with other searching schemes. TLS achieves high efficiency and effectiveness in terms of memory and time consumption, and searching accuracy.

6 Acknowledgements

This research was supported in part by the Acxiom Corporation.

7 References

[1] J. B. MacQueen. "Some Methods for classification and Analysis of Multivariate Observations"; In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, 1 : 281-297, 1967.

[2] J. C. Dunn. "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters"; *Journal of Cybernetics*, 3: 32-57, 1973.

[3] J. C. Bezdek. "*Pattern Recognition with Fuzzy Objective Function Algorithms*"; Plenum Press, New York, USA, 1981.

[4] Wikipedia. "Nearest Neighbor Search"; April 14, 2008. http://en.wikipedia.org/wiki/Nearest_neighbor_search.

[5] J. L. Bentle, J. H. Friedman, and R. A. Finkel. "An Algorithm for Finding Best Matches in Logarithmic Experted Time"; *ACM Transactions on Mathematical Software*, 3(3):209-226, 1977.

[6] R. Panigraph. "Nearest Neighbor Search Using kd-trees"; Technical report, Stanford University, 2006.

[7] 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*, 1994.

[8] D. A. White and R. Jain. "Algorithms and Strategies for Similarity Retrieval"; Technical Report VCL-96-101, University of California, 1996.

[9] J. Uhlmann. "Satisfying General Proximity/Similarity Queries with Metric Trees"; *Information Processing Letters*, 40: 4: 175-179, November 1991.

[10] L. J. Heyer, S. Kruglyak, S. Yooseph. "Exploring Expression Data: Identification and Analysis of Coexpressed Genes"; *Genome Research*, 9:1106-1115, 1999.

[11] A. Z. Broder, M. Charikar, A. M. Frieze, M. Mitzenmacher. "Min-Wise Independent Permutations"; *Journal of Computer and System Sciences*, pp. 327-336, 1998.

[12] Wikipedia. "Locality Sensitive Hashing", March 21, 2008. http://en.wikipedia.org/wiki/Locality_sensitive_hashing.

[13] D. A. Grossman, O. Frieder. "*Information Retrieval*", 2nd Edition, Springer, Netherlands, 2004.

[14] A. Andoni. "LSH Algorithm and Implementation (E2LSH)"; 2005. http://web.mit.edu/andoni/www/LSH/index.html.

[15] 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)*, 2004.

[16] P. Indyk and R. Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality"; In *Proc. Of STOC*, pp. 604-613, 1998.

[17] H. Shen, T. Li and T. Schweiger. "An Efficient Similarity Searching Scheme Based on Locality Sensitive Hashing"; *The 3rd International Conference on Digital Telecommunications (ICDT)*, Bucharest, Romania, June 29-July 5, 2008.

[18] Wikipedia, "Cluster analysis", May 4, 2008. http://en.wikipedia.org/wiki/Data_clustering.

[19] "QT (Quality Threshold) Clustering", Silicon Genetics, 2003. http://envgen.nox.ac.uk/courses/GeneSpring/GS_Mar2006/QT%20Clustering.pdf.