An Investigation on Multi-Token List Based Proximity Search in Multi-Dimensional Massive Database

Haiying Shen, Ze Li, Ting Li
Department of Computer Science and Engineering
University of Arkansas, Fayetteville, AR 72701
{ hshen, zxl008, txl005}@uark.edu

Abstract

A proximity search looks for similar complex documents such as images, sounds, DNA sequences that share two or more separately matching terms within a specified distance from within a large collection. Retrieving those similar complex documents are of great importance to many applications. To achieve an efficiency query process, many different access methods have been proposed. Token list based proximity search has been proved to be a good alternative method to the LSH for a large massive database proximity search. However, single-token based method leads to a high overhead in the results refinements process to achieve a required similarity. In this paper, we investigate how the multi-token list affects the performance of database proximity search. Numerous experiments have been conducted and the results show that two-token adjacent token list can achieve the best query performance in multi-token list based proximity search.

1. Introduction

A database of complex objects aims at storing such objects and to provide a mean to access them according to their content [1]. Besides finding exactly information, people also have great interests in looking for proximity information which contain certain key words.

For example, when a user input a sequence of keywords in the GOOGLE, the search engineer will return the results with highest similarity to the results with least similarity in an arranged sequence. In the e-bay website, in addition to returning the things the buyer desired to buy according to the inputted keywords, the search engine will also return the goods relative to the keywords which the buyer might be interested. In the bio-informatical field, since the completely identical DNA sequences are very hard to find, the scientists are also very interested in the similar DNA sequences which can be used for further research. Even in the image and speech processing field, the image and speech documents are represented by high-dimensional color histograms and a large number of their MFCC coefficients [2, 3, 4]. These data points are stored within the database in order to search the documents have some specific content [1].

To retrieval the desired results efficiently, many different access methods have been proposed. Sequential search which is a brute force search algorithm scans the data file to evaluate if the data points are within the query. Although this method is the reference for all access methods, it is not efficient in a massive database with complexity of O(n) for a single query. Other search methods [5][6][7][8][9] rely on a hierarchical partitions of the database D. This partition is provided by most of the tree-based index structures such as B-tree, Kd-tree, R-tree and LSD-tree so that the search algorithm may quickly filter out the regions that do not overlap with the query. Therefore, they are supposed to perform more efficiently than the brute-force sequential scan which read the whole data file. However, it has been shown that in high-dimensional data spaces (>20), all these methods tends to perform worse than the sequential scan due to the curse of dimensionality that is either the running time or the space requirement grows exponentially in dimension [10]. Clustering-based access methods [11, 12, 13, 14, 15] aim at clustering the database in order to group the data points and accessing only the clusters that are likely to contain the nearest neighbors of the query points. While such methods may be very efficient at query time, usually clustering methods need cluster the whole database and therefore can not be dynamically updated. Furthermore, clustering very large database has an unaffordable cost.

T. Li in [16] proposed a token list based searching scheme (TLS) that improve LSH [15] based searching

scheme. By classifying records based on token list, TLS shortens query time, reduces memory consumption and also works for different dimensional data sets. TLS stores 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. Their experiment results show that the token list scheme is more efficient than LSH-based searching scheme.

Based on the work in [16] which use one-token based token list for information searching in a massive customer data integration (CDI) , we analysis how the Multi-token based token list affects the performance of the proximity search in multi-dimensional massive database. The multi-token based token list is an ordered list in which each entry store a composed multi-token consists several key words of one record, in contrast to previous one key word per token. In this paper, n-token or multi-token based token list refers to the token list in which each entry contains a token consists of several key words of each record. The experiments show that given a certain similarity requirement, the multi-token based token list achieves a higher performance than the single token based token list. Especially, the two-token based multi-token based token list can achieve the highest query performance.

2. Relative Works

In the past few years there have been numerous studies on the problem of finding the nearest neighbor of a query points in a high-dimensional (at least three) space focusing mainly on the Euclidean space: given a database if n points in a d-dimensional space, find the nearest neighbor of a query point.

Linear search compares a query record with each record in the database once at a time. However, this method is inefficient since it leads to O(n) time latency for each query. Another approach uses the distance information to deduce k-dimensional points for records so that Vector Space Model multidimensional [5] indexing method can be used. R. F. S. Filho in [18] proposed an Omni search strategy and generalizes the concept of searching data based on some anchor points. The triangular inequality is used to limit the number of distance computation during the search. Since the base of the anchors is not defined, most access methods based on this strategy define their own anchor selection algorithm.

Bentley et al. proposed a k-dimension tree (kd-tree) data structure [6] that is essentially a hierarchical decomposition of space with long dimensions. Kd-tree is effective in a low dimensional space, but its searching performance degrades at dimensions larger than two. Panigrahy in [17] managed to improve kd-tree search algorithm that iteratively perturbs the query point and traverses the tree. Balanced Box-decomposition trees (BDD-trees) [7] are extensions of kd-trees with additional auxiliary data structures for approximate nearest neighbor searching. It recursively subdivides space into a collection of cells and measures the distance between a cell and a query point to determine whether the point to determine whether the points in the cell should be options in the kd-tree searching. These approaches map each record to a kd point and try to preserve the distances among the points. However, it is difficult to decide the value of k such that the mapping between each domain object to a k-dimensional point can accurately represent the similarity between objects.

LSH is a method of performing probabilistic dimension reduction of high-dimensional data. It is used for resolving the approximate and exact near neighbors in high-dimensional spaces. It uses a special family of locality sensitive hash functions. The main idea of the LSH is to use the hash functions to hash high-dimensional points into a number of values, such that the points close to each other in their high-dimensional space will have similar hashed values. The points are classified based on their hashed values. Consequently, the near neighbors of a query point can be retrieved by locating the points with the similar hashed values. Previous research on LSH in p-Stable distribution shows that LSH has a number of drawbacks. First, it requires a large memory to achieve fast query. Second, it needs refinement to achieve high accuracy, which leads to long processing latency. Third, LSH employs the vector model to get a record's identifier, which forms all tokens to a multi-dimensional identifier, which lead to a lot of 0s and only a few 1s. Therefore, this sparsity leads to high memory consumption and long refinement time.

T. Li [16] proposed a Token list based searching scheme (TLS) which 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 list. In searching, TLS maps a query to the token list based on the query record's token and retrieve all the record's index stored in it. Although, TLS can shorten query time, reduce memory consumption and also work for different dimensional data sets. However, since each token in the token list records the indexes of all the records that have it as keyword. For a record has a considerable number of keywords, it will take a long time in the refinement process to arrange the results in

a ordered way. Meanwhile, although the one token based token list search can retrieve all the records sharing even one common keyword, it is not likely that in the real application, user will interest in the records that share only one similarity. Therefore, in this paper, a multi-token list based searching scheme has been proposed to further reduce the query time and refinement time of one-token based searching scheme.

3. Multi-Token List Based Searching Scheme

Definition 1: A record is a string array that consists of a series of key words. Each key word is called a single-token. For example, A record

Ann Johnson | 16 | Female | 248 Dickson Street

Consists of key words (token) as "Ann Johnson", "16", "Female", "248 Dickson Street". Without specification, the single-token in this paper refer to individual keyword.

Definition2: A multi-token or n-token refer to a token consist of n keywords. For example, a 2-token based token is "Ann Johnson 16", "Ann Johnson Female" in previous example.

In the Multi-token List based searching scheme (MTLS), all the source records are parsed into single-tokens stored in a token list. These single-tokens are mutual-concatenated into n-token. Each n-token in token list contains the information about which record it belong to (the record contain it as keywords). In the target record searching process, each query record is also parsed and concatenate into n-tokens as source records do. For each n-token, the relative files are retrieved according to the token list. In order to efficiently retrieve the data, the Multi-tokens are hashed into integers and store in a hash table for efficiently n-token query and store. The data structure of the Multi-token is <hashcode, List<Record Indexes>>.

3.1 Token list construction

Since in the single token based token list search, each token is likely to retrieval a large number of records. In these retrieved records, many of them shares only one similarity with the query record which the users may not be interested. Moreover, the further refinement on the retrieved results to get rid of the overlapping information and arrange them in a specified order leads to considerable overheads. The basic idea of MTLS is to rebuild a Token List in which each entry of the list is occupied by a Multi-Token (n-token). The Multi-Token is formed by combining several keywords of each record in a permutation way to get all the possible combination of the keywords in each record. Then these multi-tokens serve as new tokens for the proximity searching.

The Multi-token (n-token) based data searching reduces the number of retrieved records by increase the similarity limitation to n. It seems that the refinement time can be greatly reduced, which lead to a sequentially decrease of query time. However, we must notice that, with the increase number of n, the permutation time to get the target multi-token will increase in the order of O(n!). Therefore, there are must be a tradeoff in the selection of token size n. In next sections, this problem will be explored.

3.1.1 One token based list construction

An example is used to explain the MTLS 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

In order to construct a one token based token list, after parsing each record into several single-tokens, these single-tokens are inserted into a token list. For some identical single tokens, only one of them will be stored in the list as a representation, but the record index stored in the rest of single tokens will be inserted into that represented one. Table 1 show an example of one token based token list.

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

Table 1: An example of one token based token list

3.1.2 Two token based list construction

For each record, we still parse it into tokens. For example, the tokens in record 1 is "Ann Johnson", "16" , "Female", "248", "Dickson Street". Then we concatenate every two of the tokens in this record and insert it into the token list, that is, "Ann Johnson 16", "Ann Johnson Female", "Ann Johnson 248 Dickson Street", "16 Female", "16, 248 Dickson Street", "Female 248 Dickson Street" and so on.

Table 2 shows an example of a two-token based token list about record 1 and record 2 in previews example.

Token	Index	Index
Ann Johnson 16	1	
16 Ann Johnson	1	
Ann Johnson Female	1	2
Female Ann Johnson	1	2
Ann Johnson 248 DS	1	
248 DS Ann Johnson	1	
16 Female	1	
Female 16	1	
16 248 DS	1	
248 DS 16	1	
Female 248 DS	1	
248 DS Female	1	
Ann Johnson 20	2	
20 Ann Johnson	2	
Ann Johnson 168 Garland	2	
168 Garland Ann Johnson	2	
20 Female	2	
Female 20	2	
20 168 Garland	2	
168 Garland 20	2	
Female 168 Garland	2	
168 Garland Female	2	

Comparing Table 1 with Table 2, we can find that although Table 2 just depicts the two-tokens of record 1 and record 2, the size of the 2-token based list is already much larger than the 1 token based list even use 4 records. It is because for a record with m key words, the number of n-token is:

$$P_m^n = m \bullet (m-1) \bullet ... (m-n+1)$$

It is intuitively that the n-token based token list will be very long. Therefore, in order to retrieve the n-token in the token list with high efficiency as single-token do, a hash table was used to map the n-token list to the hash table. That is, if for every token queries, we can go to hash table grasp it directly with time complexity of O(1). The details will be explained in section 3.2.

3.1.3 Multi-token based list construction

The construction of Multi-token based list is based on two-token based list construction. Algorithm 1 shows the pseudo-code of Multi-token based list construction

For a large amount of records need to build a token list, the computer read a record v[i] at one time and parse them into several tokens. If we want to build a n-token based token list, then we concatenate every n tokens in v[i]. If the new created n-token does not inserted into the tokenlist before, then insert it. Otherwise, drop it. After checking all the new created n-token, the computer read the next record v[i+1].

3.2 Token list Hashing Scheme

Because of the tremendous number of tokens in a massive data base, the length of the token list is very long. In order to efficiently answer these queries, i.e. in the minimum amount of time in the order of O(1), the multi-tokens are hashed into a hash table for efficiently store and retrieve.

For each multi-token t[j] in source record v[i], the multi-tokens are hashed into integers and stored in to hash table. After storing the multi-token into the hash table, its relative record index should also been linked with t_code.

3.3 Multi-token based query.

The record retrieval process is similar to the token list construction process.

For each query record, we parse it into several token. Then every n of these token are concatenated together with different order to form a n-token. For every n-token, we hash them into integers, and try to find it in the hash table. The hash table will retrieve the record indexes relative to the hash values if it contains them.

4 Performance Evaluation

We conducted experiments for the following methods:

(1) MTLS with one token, denoted as MTLS-1;
(2) MTLS with two tokens, denoted as MTLS-2;
(3) MTLS with three tokens, denoted as MTLS-3.

The testing data is synthetically generated personal data which contains many attributes such as name, gender and address. The total number of source records is 10,000. We randomly picked 97 records as query records. We compared the performance of MTLS-1, MTLS-2 and MTLS-3 in terms of sorting token time, query time, percentage of similar records in the located records and the percentage of similar records returned. The following matrices are tested:

- *Time for sorting tokens of source records*. It is the time for collecting the unique tokens of source records, and making all the combinations by picking certain number of unique tokens. Short sorting token time means that less time is spent on setting system before query start.
- *Total query time*. It is the time for retrieving the similar records of queries. It shows the efficiency of a prospecting searching method. Good similarity searching scheme should achieve fast query speed.
- *Percentage of true positives*. True positive is the located record which is similar as query. If the located record is not similar as query, it named as false positive. Efficient prospecting searching method should have high percentage of true positives.
- *Percentage of similar records returned*. It shows how much percentage of similar records in the source records can be returned. Efficient prospecting searching method should locate as many similar records as possible.

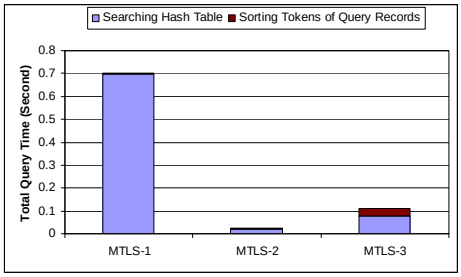

Figure 1. Total query time.

Figure 1 show the query time of different methods. From the figure we can see that MTLS-1 has longest query time, and MTLS-2 has fastest query speed than other methods. Query time consists of the time for sorting tokens of query records and time for searching hash table. The time for sorting tokens of query records is the time for getting all the unique tokens of query records, and generating the combinations of the query tokens. Time for searching hash table is the time for retrieving the similar records of query base on different combinations of the query token. From Figure 1 we can notice that sorting query tokens takes much less time than searching hash table in all the methods. Because each query record only contains about 10 tokens, it is easy to generate combination of the query tokens. However, when searching hash table according to the combination of query tokens, MTLS needs to find the location of all the records contain the combination and filter the searching results due to some records may have duplicated copies in the searching results. Because there are more records have one common token with query record than the records have two or more common tokens with query record, searching hash table time of MTLS-1 is longer than other methods. Compare MTLS-2 with MTLS-3, MTLS-2 has less searching hash table time than MTLS-3. If a record has 10 tokens, there are 90 combinations for two tokens and 720 combinations for three tokens. Therefore, MTLS-2 has to look up hash table for 90 times. MTLS-3 has to look up hash table for 720 times that is eight times more than MTLS-2, which leads to long searching hash table time. With the increase of the number of token combinations, the time for sorting query tokens increases. Therefore, MTLS-2 can achieve short query time and it is more efficient than MTLS-1 and MTLS-3. We also did experiment with different amount of query records as shown in Figure 2. The result of Figure 2 confirms that MTLS-2 achieves fastest query speed in all three methods. Because of the increases of the number of query records, the total query time is getting longer and longer. The query time increasing rates of MTLS-2 and MTLS-3 are slower than MTLS-1. When the number of query records is less than 3,000, the query time of MTLS-3 even longer than MTLS-1. Therefore, MTLS-3 only works well with large number of query records.

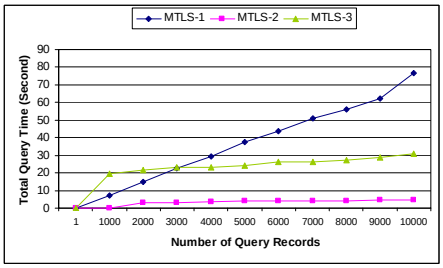

Figure 2. Total query time with different numbers of query records.

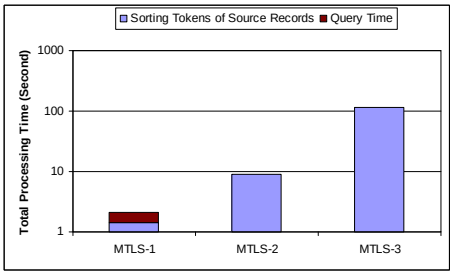

Figure 3. Total processing time.

Figure 3 presents the total processing tome of MTLS-1, MTLS-2 and MTLS-3. Total processing time includes of the time for sorting token of source records and query time. From Figure 3 we can see that sorting tokens of source records takes most of the total processing time. With the increase of the number of token combinations, the time for sorting tokens of source records increases. MTLS-3 has the longest sorting token time. If a record has ten tokens, MTLS-1 can make 10 token combinations for the record; MTLS-2 can generate 90 combinations; MTLS-3 produces 720 combinations which is the most combinations in the thress methods. Therefore, the sorting token time of MTLS-3 dramatically higher than other methods. The long sorting tokens of source records leads to long total processing time.

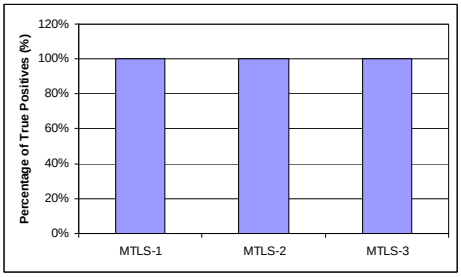

Figure 4. Percentage of true positives.

Figure 4 shows the percentage of true positives in the located records. From Figure 4 we can observe that all the records located by MTLS are similar as queries. MTLS generates all the combinations for the tokens of source records, and it assigns a unique hash code for each token combination. When searching a query, MTLS looks up the hash table base on the query token combincation. Therefore, MTLS only return the records have the same token combinations with query.

5. Conclusions

Proximity search is of great importance for many applications in our daily life. Token list method has been proved to have a better performance than Locality Sensitive Hash methods in multi-dimensional massive database. However, the single-token based token list search may lead to high overhead in the results refinements. And it is not likely that the return record sharing similarity of one to the query record will interest the user. Therefore, in order to optimize the token list based proximity search method, in this paper, we investigate how the token size affects the query time and query accuracy in the proximity search by building different multi-token lists according to multi-token size. The experience results show that two-token based token list proximity search can reach the best query performance in terms of the query accuracy and query time.

Acknowledgments

This research was supported in part by the Acxiom Corporation.

6. References

[1] N. M. Loccoz,"High-Dimensional Access Methods for Efficient Similarity Queries", *Technical Report*, Universite De GENEVE, 2005, No.05.05.

[2] T. Cover, P. Hart, "Nearest neighbor pattern classification", IEEE Trans. Information Theory IT-13(1967), pp. 609-617.

[3] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis, Journal of the Society for Information Science, 41(6), 391-407, 1990.

[4] R. Fagin, Fuzzy Queries in Multimedia Database Systems, Proc. ACM Symposium on Principles of Database Systems. 1998, pp.1-10.

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

[6] 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.

[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] 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, 1993.

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

[10] R. Panigrahy. Nearest Neighbor Search using Kd-trees. *Technical Report*, Universite of Stanford , 2006, No.04.12.

[11] K. P. Bennett, U. Fayyad, and D. Geiger. Density-based indexing for approximate nearest-neighbor queries. In Proceedings of KDD, 1999.

[12] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. IEEE Transactions on Knowledge and Data Engineering 2002

[13] S. A. Berrani, L. Amsaleg, amd P. Gros. Approximate searches: k-neighbors + precisin. In proceeding of CIKM, 2003.

[14] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In STOC 97: Proceeding of ACM symposium on Theory of computing, 1997.

[15] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In proceedings of VLDB 99.

[16] T. Li, H. Shen, and A. M. Rosequist. Token List Based Data Searching in a Multi-Dimensional Massive database. In proceeding of DMI 2008.

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

[18] R. F. S. Filho, A. J. M. Traina, J. C. Traina and C. Faloutsos. Similarity search without tears: The omni family of all-purpose access methods. In proceedings of ICDE 2001