Effectively and Efficiency Consideration for Spatial Database

Mohammed Otair, Ph.D
Amman Arab University, Amman, Jordan

ABSTRACT
Metric spaces are very useful in spatial database, and other applications that deal with it. Especially when want to found object that are similar to other object. This condition does not handle the relative position that found in some tree, such that, R-Tree, R+-Tree, and R*-Tree. Instead of handling distances between objects, using Euclidean Distance to compute the similarity between them, and retrieve the sets of object from database based on Euclidean distance that make this situation is happen and occurs. This paper proposed methods for similarity search that make the general assumption in the similarity can be done, by focusing on the effectively and efficiency in metric space for spatial database. This paper introduces how to improve the effectively and efficiency of the spatial database. It provided to extract some knowledge from spatial database has been presented. The main goal is to know how the search occurs, by using some assumption and ensures that there is similarity between a given queries Q and a set of object that found in the database.

Key words
Similarity Search, Euclidean Distance, Compression, Nearest Neighbor Query, Range Query, and Ranking.

1. INTRODUCTION
Traditional Database methods are designed to handle the relatively position between query object Q and dataset object that found in universe (U) [1]. The main problem of this implementation tasks is not straightforward (with vector and dataset method) right now, these are the lack of spatial data rules to retrieve the similarity measure. With metric space, the most common query were designed to handle exact match, partial match, range query, rank query, and join applied to some or all objects that found in database. Then retrieve the set of objects from database which similar to query object, as result it contains answer set [1, 3]. Hence, there is a need to define assessment measures for Spatial Database (SDB). Two commonly used measures are effectively and efficiency.

Effectively measures the accuracy of the result set in terms of three measures: Range, Nearest Neighbor Search, and Ranking [1, 3, and 4]. Range is defined as given a query object and radius R, return the set of objects that neighbor search to the given objects that satisfy some condition with radius (R > 0). While Nearest neighbor search is defined as given a query object and integer k, return the k objects is S according the shortest distance from Q. Ranking retrieve the set of objects with the shortest distance from given object based on some criteria.

Each type of query is difficult to handle using Quad Tree, R-Tree, and others. But queries can be handled by using metric space by using some assumption to make that happen. Efficiency measure how much fast a result is obtained and can send it from a server to a client especially in ranking measures. This can be computed by using standard algorithmic such the time complexity (e.g., the "Big Oh" notation which describe the worst case), or calculate that by using statistics methods as response time, disk I/O, etc.

Most research in SDB focus on effectively measures to retrieve the similarity with query and objects in DB. Since, it is well accepted that current systems exhibit similarity with the given object and other objects. Therefore, many papers in SDB focus only on effectively and ignoring efficiency which leads to poor effectively, efficiency is a secondary concern. For example, a recent compendium of seminal research papers [5, 6] did not include a single paper whose primary focus was efficiency. A system that takes too long to send result to clients will simply not be used.

There are many ways to improve the effectively and efficiency measures. Effectively can be improved by responding to the queries (in term of similarity retrieval) [5]. They can be summarized as follows:

- Retrieve object that fall within given range or distance from given object (Range queries).
- Retrieve objects whose value similar of given query (nearest neighbor queries) or in some time call (approximate nearest neighbor).
- Retrieve and rank the set of object base on some condition (ranking object based on stop condition).

In order to compute the distances in high dimensional space, then Euclidean Distance can be applied [3]. In the case of exact match, partial match and range queries, the time became very consuming. For instance, to calculate the distance between two points in a Z-dimensional space using Euclidean distance, it will need Z*Z-1.

When distance function is the only information in hand, then there are two choices: The first one which based on the distance of the object by deriving the *features value* [6]. For instance, given a query Q which is searching about object *G*, and the main objective is to select a value for *k* and searching about the set of *M* congruous points. Thus, the distance between objects of the query and relevant or similar *N* objects in the database will be computed using a good distance function. It will return all the objects that are very or mostly closed to the original distance function *F* for *M* objects. The second is computing the distance function that can be used to index the data with respect to

distance from few selected objects. That called distance based indexing to describe such methods [6].

The goal of distance based indexing, is to compute the distance by building an index, then finds similarity queries, and compares it with the entire dataset. Through scanning only the candidate object and ignore (pruning) the other objects that not conclude the answer set.

To improve the efficiency, the compression techniques can be used to compress the result at the server side then sending the result to the client side to decompress it there [8]. Especially in case of ranking objects with or without any stop condition, to retrieve large number of object that are similar to give an objects based on order of distance. The main goal from compression is to reduce the response time and ensure that it doesn't loss any data during sending and receiving. To achieve that the Huffman Coding can be used to compress and decompress files and prove that by algorithms that reduce the response time and the size of data from server to client, during compression and decompression stage.

The rest of this paper is organized as follows: section 2, include the problem definition. Section 3, effectively performance will be presented. Section 4, efficiency performance were described. Finally, the paper finished with concluding remarks.

2. PROBLEM DEFINITION

A metric space can be defined as follow: P = (O, L); P is the distance and O is the object domain based on the following conditions:

$L(x, y) = L(y, x)$ (Symmetry)

$L(x, y) \geq 0$, $L(x, y) = 0$, iff $x = y$ (Non-negativity)

$L(x, z) \leq L(x, y) + D(y, z)$ (Triangle inequality)

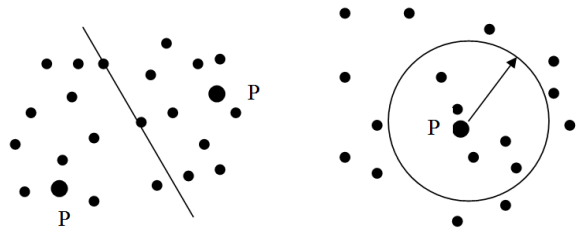
Where x, y, and z are objects in O. the similarity between objects can be measure based on the distance function L. Range query, nearest neighbor, and rank are three basic types can use in similarity measure.

3. EFFECTIVELY PERFORMANCE

3.1 Indexing Metric Space Structure

In the proposed improvement the Distance based indexing Method (Euclidean Distance) can be used, and show how to perform the similarity search. Metric trees are indexing structures for exemplary distance. They are binary trees that perform repeatedly in partitioning a data set at each node into two subsets. Two standard partitioning modes are identified Uhlmann in [5]: line partitioning and ball partitioning.

In line partitioning two object are taken from the points' set, A and B objects are selected and the data set is classified using the two selected objects. The nearest objects to each one of them will be returned, where all the objects in group A are nearer to the object A than to object B, and vice versa regard to group B as in figure 1-a. Whereas, in ball partitioning, a vantage point concept is used, where is the set of data is classified using the distances from one object [4], and the data points will be classified into two groups: the group of objects outside and inside the ball as in figure 1-b.



(a) (b)

Fig. 1 (a) Line partition (b) Ball partition

The pivot can be used to point to any class of specified objects that need to split an object during them. It also can be utilized through the process of clipping and searching other objects. Every pivot P should match the following conditions:

- $P \in S$: is an object which contains the information concerning all or some objects in *S*.
- *d (p, o)*: is an exact value of the distance *d* for all objects in *S*.
- That the value of d(p, o) falls inside some range of other values.
- Or, that the object *o* is nearer or the nearest to *p*.

The pivot can be choosing with the ball partition as following criteria:

1. Determine the threshold to partition dataset according the value given.
2. Determine the minimum value of points that fall in given threshold
3. Choose the maximum distance among points and threshold to perform the ball partition.

In this technique all or some of distance between object in database are pre-computed. Then, when implementing the queries by calculating the actual distance between the query object and some objects as initial answer set. Based on the some other initial computed distances, the remaining distances of the other objects may be evaluated.

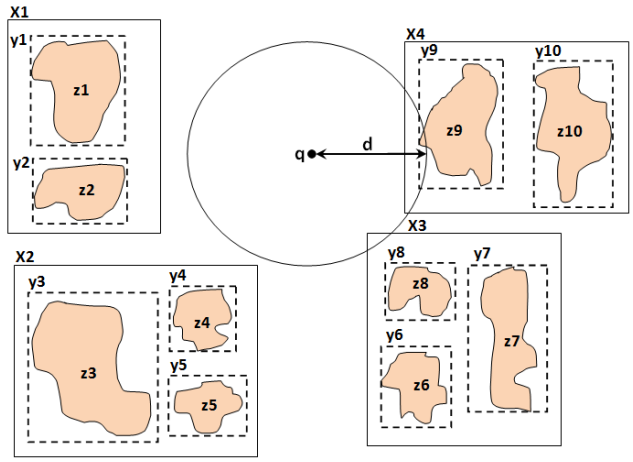
3.2 Search Structure

The search structure can be described as in the following example as in [4, 5]: There are K objects in a database D, if the object of the search process is to find an object(s) that are mostly relevant to the object of the query *q*. In other words, a metric space (D, M) can draw D and *q*, where D is a corpus (group) of objects and M is the metric distance which defines resemblance, when the distances are very small mean they are more resemblance.

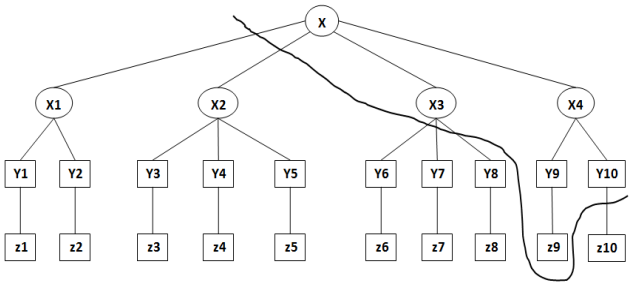
The search can be categorized with three elements type (T), type 3 indicates to bounding rectangle, type 2 indicates to node, and type 1 indicates to object. The main assumption is the Childs elements are disjoint, that each element with one parent and each element is associated distance D (Q, T). The distance from parent P to child can be used to indicate which nodes are use to search and which are pruning to avoid multiple paths during searching.

The concept can be illustrated as in the following example in Figure (2).

The nodes x (root of tree) and x1- x4 (internal node) represents the element 3, node y1-y10 represent element 2, and object z1-z10 represent element 1 that indicate to leaf node which have minimum bounding rectangle.



(a)



(b)

Fig. 2 Representation collections of two dimensional spatial objects (a) space hierarchy representation (b) sample range query representation.

The range query overlaps with X, X4, X3, Y10, Y9 and z9. This is the alternative which involves the answer set. Using the category type considered above the tree consists of three types of elements: type 1 to indicate to object z9, type 2 to indicate to bounding rectangle Y9, and Y10, and type 3 to indicate node X, X4, and X3.

The Euclidean distance techniques can be used to compute the distance from query q to the region of the query. It is used to compute three type of element, the range search can use another technique called visited and pruning. That visit only the node that intersect to query and pruning the node do not intersect to query (z1 to z8 and z10 are ignore) which are not needed to compute the distance from z1 to z8 and z10 (pruning side).

3.3 Queries and Search Algorithms

3.3.1. Range Query

In the range query, the query object denoted by q, the radius denoted by E, and the search hierarchy is T [4, 6]. During traversal visiting the set of the list is initialized to be empty. The technique of algorithm works as follow:

- The search hierarchy starting from the root, and to be sure that the root is not an object. There will be an assumption that at least two objects should available at the dataset *U*.
- Visit the children elements in the search hierarchy in order objects, which are within distance range (radius) and pruning the other object that are too far from the query *q*.
- The answer of the query obtained from an intersect between the root (*X*), the leaf region node (*Xi*) with type 2, the boundary rectangle (*Yi*) with type 1, and the object (*zi*) with type 0.
- By using the Euclidian distance metric, distances from query object *q* and all the three elements type can be measured as seen in figure 3.
- The search must compute the distance from *q* of elements that are immediately below the boundary, and if there distance much larger they are prune. So to be sure the answer set is the right one.
- The propose algorithm work in recursively way.

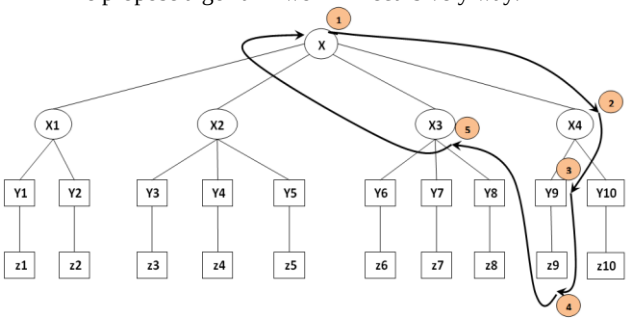


Fig. 3 range query of two dimensional spatial objects.

3.3.2. Nearest Neighbor Query

This type of queries is used to discover the number of K closest objects in space S to the object of the query [1, 4]. There are several ways to search in the nearest neighbor queries; they are depending on the method of the hierarchy of search is navigated. This paper will introduce two algorithms using two different traversal orders: Depth-first traversal, and Best-first traversal.

3.3.2.1. Depth-First Algorithm

Depth-first algorithm is an extending algorithm from range search algorithm, thus if the distance *Dk* of the *kth* nearest neighbor is know in advance, then it can be found the answer with similar range query, in order to bound the search [4,7].

The technique of the algorithms works as follows:

- Determined the value of *k*.
- *NearestList (o, k)*: this function is initially set distance to infinity (∞) until the *k* is get the *NearestList return the candidate object.*
- *ActiveBranchList(E: child element):* determined all child element of the root. So that the n is firstly visited (root), after that compute the active branch list for n which lead to n_i.
- *Sort ActiveBranchList*: which sort the *ActiveBranchList* in ascending order.

- "Branch and bound": which indicate *ActiveBranchList* is computed for each element of internal node (n_i) depends on distance of all elements related to each n inside n_i
- The algorithm is repeated recursively.
- Determine the answer set depends on *k* and return the shortest distance from final active branch list.

3.3.2.2. Best-First Algorithm

This algorithm is another version of range query that used to limits the number of branch that visited, and to speed up the answer set [4]. The algorithm works as follows:

- The *PriorityQueue*: the tree root will insert into the queue
- The *NearestList* designated to all child of *k* that inserted in it.
- The previous steps will be repeated with all elements until reach the answer set.
- The algorithm will be iterated recursively.

3.3.3. Ranking Query

The goal of ranking query is to retrieve the objects in the dataset U, in order of distance from q according to some halting condition. If the number of objects are already predefined in the result of the inquired object, that are indicated by condition of halting, then the result may be given by implementing a nearest neighbor technique to the inquired object.

The stopping condition for ranking query based on some criteria as: maximum distance or maximum cardinalities of the result. The ranking query can rank all object in the U, but the stopping condition causes only a fraction of object in U to be ranked.

Ranking query works as same as the best-first nearest neighbor algorithm, with some addition that, the distance in the nearest neighbor may have value not smaller than the value in the priority queue. So, to obtain an incremental ranking by putting object in nearest neighbor list is no larger than minimum distance in priority queue. This also leads to an unbounded size of results in setting up the position when an object becomes at the beginning of the queue, where it gets output resemblance to the subsequent object in the order of the position.

The worst case in ranking, if all objects are with the same distance, and to avoid the priority queue becomes too large. This can be accomplished by throwing away the elements (which have the largest distance) by placing the upper bound on the range to the outmost object that can be made by ranking.

4. EFFICIENCY PERFORMANCE

In the case of ranking, the objects are sorted in descending order according similarity to query object. The data that retrieved from server to client contains many objects according stop condition or not which leads to a huge data while ranking the set of object.

The main goal of efficiency is to retrieve the object that are similar to given query and to achieve that the size of data can be reduced in order to reduce the number of I\O over head, response time, and size of data during sending. This can be done by using an algorithm that can compress the candidate object and decompressed the result. Which improve the efficiency because the effectively are not introduce in this scenario (sending and receiving).

There are many algorithms that found to compress the data during sending. The paper reviews some of them and develops a new algorithm that can be used to reduce the above criteria (I\O overhead, response time, and size of data). These algorithms are as following.

4.1 Fixed and Variable Length Compression

As mention before, the set of objects in U that need ranking according the similarity of given object are stored in descending order by using array.

The fixed and variable length compression can used to compression the size of data, after convert the data to binary digits and store only the minimum number of bits using compression in the server side. The client decompresses that to get the original data and ensure that the data is preserved and there is not any loss during compression.

The fixed and variable length compression used in information retrieval is to compress the posing list. But it cannot use that while ranking object. The next section presents a new algorithms that can be use to compress the ranking data object, using Huffman coding.

4.2 Huffman Coding Index Compression

This paper proposed a new method used to compress and Decompress a file, these method convert every 8 bit block in the original file into two sub block, each sub block has 4 bits, special character can be used to represent these bits, by using Huffman tree to retrieve each sub block to show how many repeated and write the result in text file. By the way to decompress the file, Huffman table or tree can be used to retrieve the original file. Our improvement saves 35% as an average of the size of the original file.

4.2.1 Methodology

This method assumes that every 8-bit block of input file as one block, and divide it into two sub block, with each 4 bit per sub block, in order to distinguish the stream of bits star(*) can be used between each 8-bit. Then, these bits can be replaced by its substitute characters as in the following table.

Table 1. Representative bits with specified character

AA	AB	AC	AD	BA	BB	BC	BD	CA	CB	CC	CD	DA	DB	DC	DD
0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

Table (1) shows all possible cases for 4 characters (A, B, C, D). For example: if the first bit is 0100 then replace it with BA. This algorithm can be used for binary files that also including a text files.
See appendix for more details about our algorithms that used for compression and decompression the file which contain N-Bits.

4.2.2 for Examples that Display this Method (Compression and Decompression)

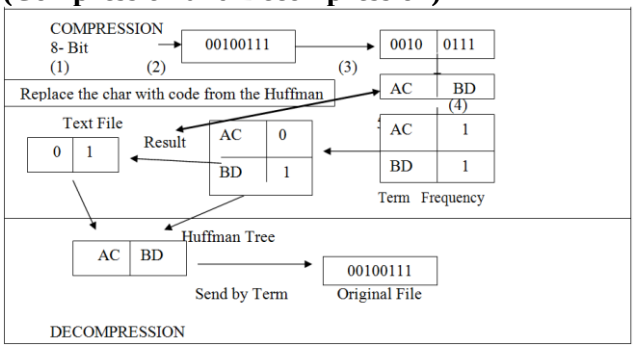


Fig. 4 The compression and decompression for the text file that contain 8 bit.

5. CONCLUSION

This paper described that variety of approaches can be improved effectively and efficiency in Spatial Database Indexing. Without efficiency, users will simply not take advantage to find the sets of objects that similar to query object. The proposed algorithms save more than 35% of original size, fast response time, and reduce I/O overhead. Firstly, the effectively in Spatial Database indexing is described, and present how can improve the effectively of query that are presented above. After that some technique were introduced which can be used to compression the data and reduce the size of it by using Hoffman Coding and present algorithm with detail in the appendix. Finally the compression file use speed the response time of ranking object.

6. REFERENCES

[1] Eamonn Keogh, Selina Chu, and Michael Pazzani (2001): A New Approach to Indexing Large Databases

[2] Slobodan Rasetic, Jörg Sander, James Elding Mario A. Nascimento (2005), A Trajectory Splitting Model for Efficient Spatio-Temporal Indexing

[3] Ralf Hartmut GQting (2004),An Introduction to Spatial Database Systems. Received July 25, 1994; accepted August 18, 1994.

[4] TOLGA BOZKAYA, and MERAL OZSOYOGLU (2000). Indexing Large Metric Spaces for Similarity Search Queries

[5] ESSAM A. EL-KWAE , and MANSUR R. KABUKA (2000). Efficient Content-Based Indexing of Large Image Databases.

[6] GISLI R. HJALTASON and HANAN SAMET (2003). Index-Driven Similarity Search in Metric Spaces . ACM Transactions on Database Systems, Vol. 28, No. 4, December 2003, Pages 517–580.

[7] Alock Aggarwal and Jeffery scott Vitter. The input/output complexity of sorting and related problem. Communication of ACM,1988.

[8] Ravidra K. Ahuja, Thomas L. Ma and James B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, 1993.

[9] http://en.wikipedia.org/wiki/Euclidean_distance,Elena Deza & Michel Marie Deza (2009) Encyclopedia of Distances, page 94, Springer

-Appendix A-

That includes the detail of the Algorithms that used to compress the file and decompressed it, and the Euclidian Distance Equations.

Algorithm Compression File (input original File, A [ ], K [2, 16], T[2,17])

1. Read the Original File

I=0

While (there are more block in the file) do

{

For each 8 bit do the following

{Divide the block with 2 sub block each sub block contain 4-Bit

A [I+1] = sub block [1, 2, .., Ithblock]

// each sub block contain 4 bit

}

// divide the block with 2-sub block each sub block contain 4 bits separately

// each element in the temp array contain 4 bit and separate between 8 bit with

Stars that it used when decompressed the file.

A [I+1] = '*'

// used comma to separate between each 8 bit (Separate between blocks) in the temp array

}

2. Procedure Convert_Block_To_Chars (A [ ], K [2, 16])

// the input to this procedure is temp array A [ ] and 2-dimensioan array K[2,16] that contain all possible of the character

The output is replace each bit in the temp array (A [ ]) with the special characters that occur in the 2 dim. Array (I.E call by reference). Separated between each eight bit in the array A [ ] with Star

3. Procedure Find_Frequency (A [ ], T [2, 17])

// this procedure calculate the frequency for each character by declare 2 dimensional

Array with 2 rows and 17 columns first row contain all possible of the characters and second row represent the counter for each character, and the 16 columns for possible characters and 1 to the star.

4. Apply Huffman tree to array A [ ] and associate code for each term in the array

5. Create the temp file (text file) that used to write the results (temporary array A[ ] by take the associate code from the Huffman tree for each element in the array. also the Huffman tree or the Huffman table that use during decompressed the file.

Display the working of the procedure that use in the Algorithms

```
Procedure Convert_Block_To_Chars (A [ ], K [2, 16])
 {For j=1 to length [A]
 I=0
   Found = true
  While (found) and (I<= 16) then
   {     I=I+1
          If ( A[j] = K [2, I])   then
             A[j] = K [1, I]
             Found = false
    }
```

```
  }
Procedure Find Frequency (A [ ], T [2 , 17])
 {For j=1 to 17
   C=0
   For I=1 to length [A]
     If T [1, j] =A[I] then
          T [2, j] = C+1
 }
```

Algorithm Decompression File (input Compressed File, Huffman table)

Create empty temporary file that will used to write the decompression file Use the loop statement that to replace each associate code in the compressed file with the characters from the Huffman table or tree.

Use another loop statement to replace each characters in the temp file with the associate code that show in the figure 1 to retrieve the original file if appear the star in the temp file then separate each block to another block until to retrieve the original file.

Euclidean distance

The **Euclidean distance** between two points $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_1, q_2, \ldots, q_n)$, in Euclidean *n*-space, is defined as:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

One-dimensional distance

For two 1D points, $P = (p_x)$ and $Q = (q_x)$, the distance is computed as:

$$\sqrt{(p_x - q_x)^2} = |p_x - q_x|$$

Two-dimensional distance

For two 2D points, $P = (p_x, p_y)$ and $Q = (q_x, q_y)$, the distance is computed as:

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Alternatively, expressed in circular coordinates (also known as polar coordinates), using $P = (r_1, \theta_1)$ and $Q = (r_2, \theta_2)$, the distance can be computed as:

$$\sqrt{r_1^2 + r_2^2 - r_1 r_2 cos(\theta_1 - \theta_2)}$$

Mohammed A. Otair is an Associate Professor in Computer Information Systems, at Amman Arab University-Jordan. He received his B.Sc. in Computer Science from IU-Jordan and his M.Sc. and Ph.D. in 2000, 2004, respectively, from the Department of Computer Information Systems-Arab Academy. His major interests are Mobile Computing, Data Mining and Databases Neural Network Learning Paradigms, Web-computing, E-Learning. He has more than 30 publications.