Abstract:
The traditional method of comparing values against database might work well for small database but when the database size increases invariably then the efficiency takes a hit. For instance, image plagiarism detection software has large amount of images stored in the database, when we compare the hash values of input query image against that of database then the comparison may consume a good amount of time. For this reason we introduce the concept of using metric trees which makes use metric space properties for indexing in databases. In this paper, we have provided the working of various nearest neighbor search trees for speeding up this task. Various spatial partition and indexing techniques are explained in detail and their performance in real-time world is provided.

Keywords: Metric Trees, Metric Space, Nearest Neighbor Search Trees, Spatial Partition, Indexing Technique

Introduction to Metric Tree/ Nearest Neighbor Search Trees

With the ever-growing multimedia (audio, video, images, etc) there is an urge to manage it. In order to retrieve information from the database the brute force method proves to be costly in every aspect while retrieving. The reason is that they need to cater with how a particular multimedia file is similar to other and the similarity factors to be considered while comparing are image patterns, texture, color, sound, shape [1]. Therefore we make use of distance function to account for similarity. Thus to increase the efficiency of executing similar queries we need to have right indexing technique. It is for this reason we make use of metric trees. Metric trees use the properties of metric spaces which are positivity, symmetry and triangle inequality. When we talk about these properties we consider the absolute distance between the entities in question (say two points) [5]. The explanation of the properties is:-

1. Positivity
\[ d(a, b) \equiv |a - b| \geq 0 \]

2. Symmetry
\[ d(a, b) = |a - b| = |b - a| = d(b, a) \]

3. Triangle Inequality
\[ d(a, b) = |a - b| = |a - c + c - b| \leq |a - c| + |c - b| \equiv d(a, c) + d(c, b) \]

All indexing techniques primarily exploit Triangle Inequality property. We’ll discuss four of these indexing techniques BK tree, VP tree, KD tree and Quad tree in detail [5].

BK-tree

Named after Burkhard-Keller (BK tree). Many powerful search engines make use of this tree reason being it is fuzzy search algorithm. The problem definition for this type of tree can be: “Find in the dictionary, of some finite size, accounting for all the string that match given word, taking into account k possible differences.” In simpler terms we mean that if there is a string of hash values having a finite length that has to be matched from given data set then at the most “k” error terms (differences) are allowed. For example if your input query has requested for “sunny” with 1 possible error term then “suny” or “bunny” are acceptable and so on. Instead of using the brute force of searching throughout the database which would take O(n*m^2) where m is the average length and n is the word count, we try to find the nodes that are within the admissible error range and consider only those nodes. Next we need to measure distance of the input query with entries in database in order to understand how many error terms are there and retrieve only those entries which satisfy the error criteria defined [3].

Levenshtein Distance

Levenshtein distance is used in BK tree implementation, it is a metric used to find difference between strings in BK tree. It satisfies as a metric space. Levenshtein Distance basically tells us how much modifications are required for let’s say string1 to represent in string2. Playing can be
represented as Praying by removing R and appending L, hence the Levenshtein Distance is 1.

**Praying**

**Playing**

Similarly, if we have two binary strings then instead of alphabets we have to replace 0’s and 1’s. For example, d (1000001000, 0000000000) is 2.

**Example of BK tree**

We create a tree based on the Levenshtein Distance, below is a tree having Delhi as its root node and leaf nodes as Mumbai, Seoul and London and so on and weights represent the Levenshtein Distance.

Let’s say if the query requires information with "Japn" as string. We need to find the string "Japn" from the given tree with the admissible error being (n) =1. We start with the root node and calculate Levenshtein distance with our input string.

\[ d(\text{Japn}, \text{Delhi}) = 5 \]

Then we assign a range that specifies the nodes to be explored for which we have a lower limit and an upper limit. Limits are defined with formula as lower limit=\(d-n\) and upper limit=\(d+n\) (for our example 4 to 6). Hence nodes having 4, 5 and 6 will be explored and we’ll calculate Levenshtein distance again till the condition that the distance between the data present in database and input query (Japn) should be less than or equal to admissible error (n).

Hence, 4 is lower limit and 6 is higher limit. We repeat this procedure till the last node.

\[ d(\text{Japn}, \text{Seoul}) = 4 \]
\[ d(\text{Japn}, \text{Mumbai}) = 6 \]
\[ d(\text{Japn}, \text{London}) = 5 \]

Considering Seoul as parent and exploring it’s child
\[ d(\text{Japn}, \text{Japan}) = 1 \]

Even though if we find distance to be equal to n we have to search remaining branches as priority should be given to Levenshtein distance equal to zero. Assigning the admissible error should be optimal generally admissible error of 2 proves to be working fine but as we increase the performance starts degrading.

**Performance of BK tree:**

The database size signifies the amount of data stored and strings in the range of about 100 were searched with the length of the string to be searched being 5-10. We have two graphs one showing the nodes scanned for a particular string and the other nodes found with matching string. One important note, string being searched for this particular graph had admissible error = 1. We find the difference between the scanned percentage nodes and found percentage nodes, it should be as low as possible ideally 0. As seen from both the charts as the
database size increases the number of irrelevant node visited increases

**VP Trees (Vantage Point)**

The VP-tree divides the searching process into two regions left and right sub tree depending on the vantage point thereby reducing the computation. This vantage point is nothing but a randomly selected point or there are other way to select the vantage point (no of nodes, distance to other nodes, etc) but they require computation and thereby incur additional cost. The implementation of this technique has a very general method and a simple logic. The distance from each point to VP is computed, and then after having all the values we calculate median value for it. The points are sorted depending on whether they are less than or greater than the median value and the two sub trees are formed. We select all points less than median to left sub tree and greater to right sub tree. We recursively follow the same procedure for each node. This median distance acts as a separating constraint. Hence we can think of this median distance as a radius and all the values less than median distance (left sub tree points) fall inside the circle and all the values greater than radius (right sub tree points) fall outside the circle.

The data structure implementation of VP tree contains VP Id, median distance and address of next left and right sub tree node [11]. The figure below explains the concept

```
Here, V – Vantage Point

d(V,P) > median distance
d(V,R) > median distance
d(V,Q) > median distance
```

```
d(V,X) < median distance
d(V,Y) < median distance
```

```
d(V,Z) < median distance
```

Points X, Y and Z form the left sub tree
Points P, Q and R form the right sub tree
Hence we totally 'prune' one-half of tree and continue pruning till we get our goal node thus dealing one subtree at a time. This pruning helps while retrieving information from the database thereby increasing the rate of retrieving information. But this act of pruning comes at the cost of calculating median and comparing the median with all points.

**Performance of VP tree:**

The graph has x-axis as size of the database and on y-axis the average number of nodes visited as we know that in VP tree we have decision functionality which is why we make very few searches. As database size goes on increasing, search time shows logarithmic growth. VP tree performs best for a large database. The creating of the tree takes O(nlogn) where n is the number of nodes and searching time complexity varies but an efficient technique should have O(logn) as expected time [2].

**Kd-tree (K-Dimensional)**

Kd tree is spatial data partition tree. It is also called as K-Dimensional Tree. Kd tree construction is a Binary Search tree where data value stored in each node is k dimensional point in space. Kd trees are used in nearest neighbor search and other several applications. The initial step in inserting new node is traversing kd tree from root node as starting node and move to either left or right based on the data value which compares the value in ‘k’th dimension.

If we find the node under which the data value should be placed then we can add new data on either right or left depending upon the data value and comparison with node under which data value should be placed [5].
To find whether a point is present in left or right subtree?

Here we consider a two dimensional tree. First align the root node in X plane, then right and left tree will contain all the points whose coordinates in X plane are greater than equal to that of root node and less than that of root node respectively. Here (4,7) represents root node which is aligned in X-plane. Now the coordinate (3,8) has smaller x-coordinate than the root node hence placed in left tree whereas coordinate (18,16) is placed in right tree since it has x-coordinate greater than that of coordinates (4,7). [5]

Kd tree Example

![Kd tree Example Diagram]

Performance of kd-tree:

Performance of kd-trees can be measured in terms of number of queries Q and time for processing each queries T. The Figure 1.1 shows relationship between Q AND T. From the Figure we can conclude that query time T grows approximately linearly with respect to number of queries Q. Approximate time complexity is O(nlogn). [6] The second graph shows the relationship between database size and average number of nodes visited. As database size goes on increasing, search time also shows logarithmic growth. It is quite evident that the performance of kd tree is less efficient than VP tree although after increasing the database size the curve looks similar to that of VP tree.[11]

![Performance Of Kd trees](chart)

Search Cost vs Database Size

![Search Cost vs Database Size](chart)

The Quadtree

A quadtree is a spatial data partition tree. It is used to find nearest neighbor distance and in optimizations of various algorithms. The root has exactly four child nodes hence it is called as quadtree. Root node covers the entire portion of space whereas other node covers some specific portion of space. We can insert new data into quadtree easily. In quadtree each node is also divided into four nodes as they now act as root node for newly generated nodes. This methodology of dividing the single nodes into four nodes is used in two dimensional searches. Quadtree is classified into different types on the basis of the data it contains i.e. points, curves, lines, shapes and areas. [8]

4.1 Quadtree example

In the below example first we start with single root node A as shown in figure. Now divide this root node A into four small squares as A1, A2, A3 and A4. These small squares are termed as child node.
The graph(a) shows query optimization i.e. relationship between query selectivity percentage and response time in second. The graph(b) shows average response time taken for queries on two dimensional database i.e. relationship between database size and response time in second. The database size is approximately 57,000 million polygons whereas the database size in graph(b) is approximately 1.5 million polygons. Samet and webber represented polygon maps using point-region quadtree. Polygon map is used to store each vertex or points of polygon map in point-region quadtree. We can increase the size of this polygon map by cloning the polygon data. [11]

**Conclusion**

In this work, we have provided the working of BK tree, VP Tree, KD tree and Quad tree. These techniques have been tested and it has been found that VP tree proves to be forerunner out of the 4 techniques followed by Kd tree. For future work we
suggest to create a combination of two or more techniques making use of hybrid indexing method. In our opinion VP tree works best in all cases as compared to other trees. But it has been found that as the database size increases eventually the performance of Kd tree and VP tree is found to be same.

REFERENCES: