MOBHRG: FAST K-NEAREST-NEIGHBOR SEARCH BY OVERLAP REDUCTION OF HYPERSPHERICAL REGIONS

Omar U. Florez[a], Xiaojun Qi[a], and Alexander Ocsa[b]

Omar.Florez@usu.edu, Xiaojun.Qi@usu.edu, and aocsa@ieee.org
[a]Computer Science Department, Utah State University, Logan, UT 84322-4205
[b]Computer Science Department, San Agustin University, Arequipa, Peru

ABSTRACT

We propose a minimum overlap based hyperspherical region graph indexing structure to achieve fast similarity-based queries for both low and high dimensional datasets. Specifically, we reduce the region overlaps in the graph construction phase by incrementally dividing each saturated hyperspherical region and removing the longest edge of a minimum spanning tree representation of the internal objects. This overlap reduction scheme creates more separated regions, so fewer regions as potential paths are traversed when a query is issued. We also introduce a *k*-nearest-neighbor search scheme by automatically deciding the search radius to return the required number of nearest neighbors. Our extensive experimental results show the effectiveness of the proposed indexing structure compared with other tree and graph based indexing structures.

Index Terms— Hyperspherical region graph, minimum spanning tree, overlap reduction, *k*-nearest-neighbor search.

1. INTRODUCTION

Searching desired information from a large-scale database plays an important role in daily activities. Indexing techniques are viable solutions to speed-up the search operations. Various data structures have been proposed to index vector representations of objects in the Euclidean space. However, the metric space is more common when the objects are images, videos, CAD drawings, XML data, DNA sequences, or time series. Therefore, the research trend is to develop efficient indexing techniques for high-dimensional metric data, where the distance function varies with the data types. However, this is a challenging task mainly due to the following factors: 1) Distance functions for high dimensional data are expensive. 2) The search space tends to be more uniform in terms of the density when the dimension increases. 3) It is more difficult to prune data during the search process. Here, we briefly review several representative indexing techniques.

Tree-based metric indexing structures, including M-tree [1], Slim-tree [2], DF-Tree [3], PM-Tree [4], and DBM-Tree [5], are based on regions of fixed sizes. They directly implement each region as a page in the secondary storage. They are proven to be suitable for low and medium dimensional datasets. However, high dimensionality makes most regions overlap with each other. Therefore, the similarity search algorithms need to consider more regions as potential paths to be traversed when a query is issued. SA-Tree [6] is a tree-based indexing structure that does not use fixed-size regions. It approximates the Voronoi representation of the database objects through distance-based hierarchical relationships. It has shown fast response times for high dimensional datasets. Graph-based metric indexing structures, such as RNG (Relative Neighborhood Graph) [7] and HRG (Hyperspherical Region Graph) [8], reduce the overlap between regions using the proximity between objects. They implement each HR (Hyperspherical Region) to ensure that two adjacent objects are neighbors in the region only if there is not any other object in their neighborhood. The HRG indexing structure further makes the representative objects as vertices and region centers to facilitate the search process.

In this paper, we present MOBHRG (Minimum Overlap Based HRG) indexing structure, an improvement over HRG, by explicitly reducing the region overlaps. Specifically, we introduce a new construction method to incrementally divide each saturated HR by removing the longest edge of a MST (Minimum Spanning Tree) representation of the internal objects. This construction minimizes the overlap degree of the resultant HRs. We also introduce a *k*-NN (*k*-Nearest-Neighbor) search scheme by automatically deciding the search radius to return the required number of nearest neighbors. Our experimental results demonstrate that the overlap reduction facilitates the search for low and high dimensional data and achieves faster construction times. Our proposed overlap reduction scheme can also be applied to other indexing structures to obtain faster search times in high dimensional datasets. The rest of this paper is organized as follows. Section 2 presents our proposed indexing structure in terms of construction and *k*-NN search. Section 3 presents the experimental results to demonstrate the effectiveness of our approach. Section 4 concludes the paper.

2. THE PROPOSED INDEXING STRUCTURE

We propose to build a special graph structure, MOBHRG, to facilitate high dimensional similarity queries in the metric space. The objects in the database are stored in HRs. MOBHRG is a dynamic structure, capable of inserting new objects or deleting existing objects at any time with the minimum updating cost. This dynamic construction ensures the minimum overlaps among HRs by removing the MST-based longest edge when each HR reaches its full capacity. Our proposed MOBHRG improves the *k*-NN-based query performance since the overlap reduction leads to fewer regions as potential paths to be traversed when a query is issued. In the following subsections, we explain the two major components in detail.

2.1. The Construction of MOBHRG

The basic idea of constructing MOBHRG is as follows: When inserting a new object x into MOBHRG, we identify an HR, which is the closest to x, to perform the update process. If the HR is not saturated (i.e., the number of objects in HR, |HR|, is less than the highest capacity c) and the distance from x to the representative object (the center) a_{opt} of the chosen HR is less than or equal to the radius of HR, we insert x into this HR. If the HR is saturated and the distance from x to a_{opt} is less than or equal to the radius of HR, we insert x into this HR and split the saturated HR into two HRs by removing the longest edge in the MST representation of the objects in the saturated HR. The centers of the two split HRs are updated, respectively. For the HR which contains multiple objects, we choose its center as the object that is closest to the centroid of the HR and its radius as the distance from the center to the farthest object in the HR. For the HR which contains one object v_2, we perform the range search using v_2 as the center of a query to rebuild a HR to cover a set of objects around v_2 (i.e., calling **LocalInsert** function). If the distance from x to a_{opt} is larger than the radius of HR, we perform the range search using x as the query center to build a HR to cover a set of objects around x (i.e., calling **LocalInsert** function). This update process minimizes the number of vertices visited during the insertion. Here, the feature vector of the original data set corresponds to the vertex in the graph. Fig. 1 shows the algorithmic view of building a MOBHRG.

The algorithmic view of **LocalInsert** function is summarized in Fig. 2. Here, ε ranges from 0 to 1 and is a tolerance parameter to scale the estimated radius for the range query. A large ε leads to a high cost edge update and a small ε leads to a more cost effective edge update with suboptimal graph structures. By using the range query with distance r' as the radius, we obtain a set of vertices in the graph which are potentially neighbors to the new vertex c_2. As a result, the algorithm only updates the neighborhood relationships of this set of vertices.

```
Function buildMOBHRG(objects in the database)
1.  Randomly choose the order of the objects in the
    database (i.e., the vertices in the graph) {$A_1$, $A_2$, .., $A_n$}
    for constructing the MOBHRG.
2.  Start with the first vertex $A_1$. Set the capacity of its HR
    (i.e., $|HR(A_1)|$) to 1 and set $A_1$ as the center of the HR.
3.  For each vertex $x$ sequentially selected from {$A_2$, ..,$A_n$},
    perform the following operations:
    3.1. Find the center of each HR. For each center $a$:
         1) Find its neighboring vertices $N(a)=\{a_1,\ldots,a_k\}$.
         2) Compute the traverse distance from the new
            vertex $x$ to each vertex in $N(a)$.
         3) Save the shortest traverse distance in $d(a, x)$.
    3.2. Find $HR(a_{opt})$, which has the shortest traverse
         distance to $x$, based on all the $d(a, x)$'s.
    3.3. If $d(a_{opt}, x) \le Radius(HR(a_{opt}))$ and $|HR(a_{opt})| < c$
            $HR(a_{opt}) = HR(a_{opt}) \cup x$;
            $|HR(a_{opt})| = |HR(a_{opt})| + 1$ ;
            Save distance $d(a_{opt}, x)$;
    3.4. Elseif $d(a_{opt}, x) \le Radius(HR(a_{opt}))$ and $|HR(a_{opt})| = c$
         1) $HR(a_{opt}) = HR(a_{opt}) \cup x$;
         2) Build the MST $T$ using the vertices in $HR(a_{opt})$
         3) Remove the longest edge, which connects two
            vertices $v_1$ and $v_2$ in $T$, to obtain two HRs.
            Here, we denote $v_2$ as the isolated vertex after
            the edge removal.
         4) Update the center of the current HR, where the
            new object $x$ is added, as the vertex closest to
            the centroid of the HR. Here we denote this
            center as $a_c$ and its HR as $HR(a_c)$.
         5) Call LocalInsert($a_c$, $v_2$) to create the HR for $v_2$.
    3.5. Elseif $d(a_{opt}, x) > Radius(HR(a_{opt}))$
            Call LocalInsert($a_{opt}$, $x$) to create the HR for $x$.
```

Fig. 1: Algorithmic view of function **buildMOBHRG**.

```
Function LocalInsert (Vertex $c_1$, Vertex $c_2$)
1. Find $v_1$, which is the nearest vertex to $c_2$ on the graph;
2. Find $v_2$, which is the farthest neighbor of $v_1$;
3. Set the radius $r'$ as
      $\max\{d(c_1, c_2), d(v_1,c_2)+d(v_1, v_2)\}*(1+\varepsilon)\}$;
4. Perform the range query using $c_2$ as the center of a query
   and $r'$ as the radius.
5. Result = Range query results $\cup$ $c_2$;
6. buildMOBHRG(Result); //here Result is a set of vertices
```

Fig. 2: Algorithmic view of function **LocalInsert**

Fig. 3 illustrates the step-by-step construction of MOBHRG with a capacity c=4 for all HRs. Each object (vertex) is labeled according to the insertion order while constructing MOBHRG. Here, we mark the center of an HR using a large gray circle and mark the remaining vertices within an HR using smaller solid black circles. The longest edge in MST is marked by a red line. A bold black solid line is used to connect a pair of HRs.

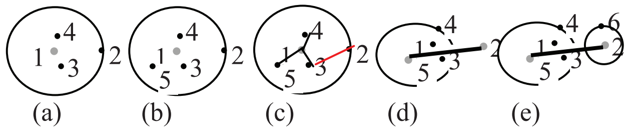


Fig. 3: Illustration of step-by-step MOBHRG construction. (a) Insert 1, 2, 3, and 4 into HR(1). (b) Insert 5 into HR(1). (c) MST of internal vertices in HR(1). (d) Split HR(1) into HR(5) and HR(2). (e) Insert 6 into HR(2).

2.2. The K-Nearest Neighbor Search

A naïve approach in performing the *k*-NN search is a linear search process. It is slow for a large scale database of high-dimensional objects. Therefore, we propose to minimize the number of distance computations by performing a few range queries with dynamically decreasing radii. A range query is to retrieve the objects from the database that are at most at a radius *r* from the query object *q*. Here, we aim to find the optimal radius *r* that covers the *k* nearest objects of *q*. Specifically, we reduce the search space by only visiting HRs that are close enough to *q*. We first compute the distance between *q* and the center of each HR. The distances from the vertices *v*'s within each HR to *q* are computed only when $|d(q, center) - d(center, v)| < rangeK$ (i.e., $d(q, v) < |d(q, center) - d(center, v)| < rangeK$ based on the triangular inequality), where *rangeK* is the search radius of *q* and is initialized as positive infinitive. A new search radius *rangeK* is dynamically reduced based on two factors: the minimum distance of all d(*q*, *v*)'s and the current *rangeK*. If more than *k* objects are returned by performing the range search with a radius of *rangeK*, we recursively repeat the process until *rangeK* is small enough to find *k* nearest neighbors. Fig. 4 summarizes the algorithmic flow of our proposed *k*-NN search.

Fig. 5 demonstrates the search paths, shown in orange color, to find 3 nearest neighbors using SA-tree, RNG, and MOBHRG, respectively. It clearly shows that our proposed indexing scheme computes fewer distances due to its more compact representation and non-hierarchical structures.

```
Function KnnSearch(Vertex query, int k, graph MOBHRG, float rangeK)
1.  Compute the distance from query to the representative
    object (i.e., the center) of each HR in the MOBHRG
2.  Order the HRs {$HR_1$, …, $HR_n$} in the ascending order.
3.  For each HR, find all the vertices v's within the HR,
    that satisfy the following condition:
      $|d(query, center) - d(center, v)| \le rangeK$
    where rangeK denotes the search radius of query and is
    specified as a parameter of KnnSearch function, center
    represents the center of the HR holding v, d(query,
    center) is computed from step 1, and d(center, v) is
    computed during the MOBHRG construction.
4.  For each vertex v found in step 3, compute the distance
    from query to v, d(query, v).
5.  Set min_dist as the minimum value of all the d(query,
    v)'s found in step 4).
6.  Update the search radius of query by:
        $rangeK = \min(min\_dist, RangeK)$ ;
7.  Keep the vertices sv's that satisfy the following
    condition: $d(query, sv) \le min\_dist + 2\times rangeK$.
8.  If the number of vertices sv's (i.e, |sv|) is larger than k,
        8.1. Reduce the graph structure to G1 using all the
             vertices sv's found in step 7.
        8.2. Call KnnSearch(query, k, G1, rangeK)
9.  Elseif $|sv| \le k$,
        9.1. Order sv's in an ascending order based on
             d(query, sv).
        9.2. Return top k sv's as the search results.
```

Fig. 4: Algorithmic view of function **KnnSearch**.

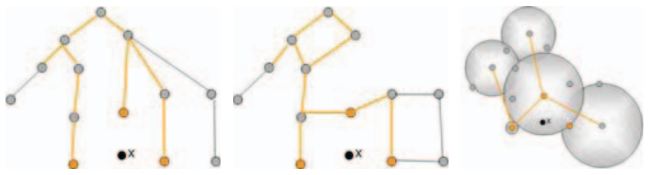


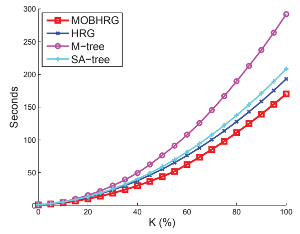
Fig. 5: Comparison of a *k*-NN (*k*=3) query in three indexing structures. Left: SA-tree; Middle: RNG; Right: MOBHRG.

3. EXPERIMENTAL RESULTS

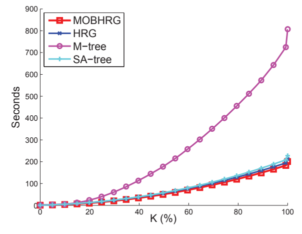
We compared the proposed MOBHRG indexing technique with SA-tree [6], M-tree [1], and HRG [8] in terms of the construction time, the *k*-NN query response time, and the overlap degree using two real and two synthetic datasets. One real dataset consists of 6,000 214-D objects (features) obtained from the COREL image database and the other real dataset consists of 6,000 36-D objects obtained from the COREL image database. One synthetic dataset consists of 1,500 16-D vectors normally distributed in 10 clusters with the standard deviation of 0.1 within a unit hypercube. The other synthetic dataset contains 1,000 2-D vectors normally distributed in 10 clusters with the standard deviation of 0.1 within a unit square.

Fig. 6 compares the construction time of four structures by incrementally adding objects from four datasets, respectively. The K(%) on the *x*-axis indicates the percentage of all the objects in each dataset is added during the construction. It shows that SA-tree always takes less construction time than MOBHRG for high dimensional data due to more edges involved in MOBHRG. MBOHRG always takes less construction time than M-tree since its overlap reduction scheme can quickly locate the region to be updated. MBOHRG also takes more construction time than HRG in the 216-D dataset mainly due to the overlap reduction process.

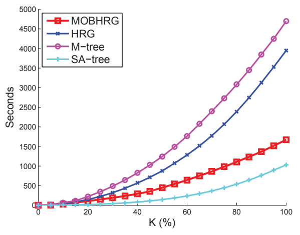
Fig. 7 compares the query response time of four techniques on four datasets with different *k*'s. The K(%) on the *x*-axis indicates the percentage (5% to 100%) of the total number of objects in each dataset, that is used as *k*. We randomly choose all the objects in each dataset as a query to perform *k*-NN search. The *y*-axis shows the average query response time. It shows our structure achieves the fastest response time for two real datasets. It performs better than two tree-based techniques and achieves comparable performance as HRG for two synthetic datasets. This is mainly because the longest edge of the MST often is the farthest objects in the HR in lower dimensions.



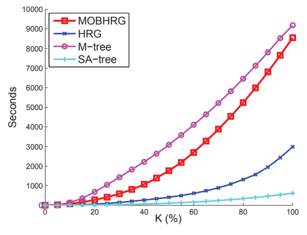
(a) Synthetic 2-D database



(b) Synthetic 16-D database

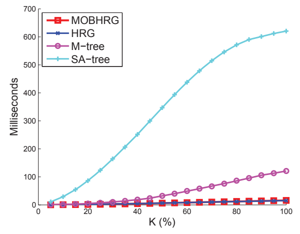


(c) Real 36-D database

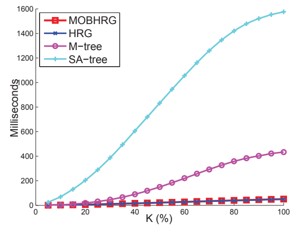


(d) Real 214-D database

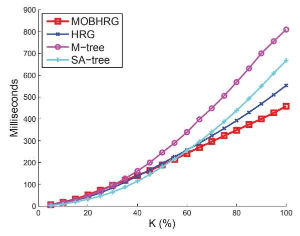
Fig. 6: Comparison of indexing structure construction time.



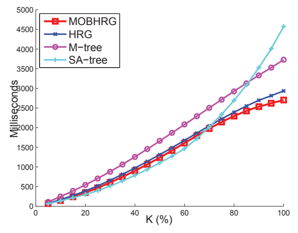
(a) Synthetic 2-D database



(b) Synthetic 16-D database



(c) Real 36-D database



(d) Real 214-D database

Fig. 7: Comparison of query response time.

Fig. 8 compares the overlap degree of HRG and MOBHRG in terms of the number of separated regions. We quantify the overlap degree between two HRs, O(HR_1, HR_2), as the ratio of the distance of their centers to the sum of their radius. Two HRs do not overlap if O(HR_1, HR_2)>1. The total overlap degree is computed as the sum of overlap degree between all pairs of HRs normalized by the total number of HRs. It measures the overall number of separate regions in each structure. Higher value means fewer overlapped regions. It shows our structure results in more separate regions, especially for higher dimensional datasets.

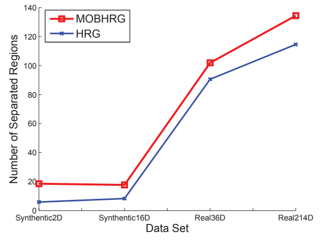


Fig. 8: Comparison of the overlap degree.

We also computed the query accuracy of our indexing structure using the linear search results as the ground truth. Our structure achieves 100% accuracy for both synthetic datasets. For top 20 and top 25 returns (*k*=20 and 25), the query accuracy for real 36-D dataset is 92.95% and 92.65% and the query accuracy for real 214-D dataset is 100% and 99.98%, respectively. The search time is around 50%, 40%, 50%, and 70% of the linear search time for synthetic 2-D, synthetic 16-D, real 36-D, and real 214-D, respectively.

4. CONCLUSIONS

We propose a new graph-based data structure to index objects in the metric space by reducing the overlaps among HRs. Specifically, we introduce a new construction method to incrementally divide each saturated HR by removing the longest edge of a MST representation of the internal objects. We also introduce a *k*-NN search scheme by automatically deciding the search radius to return the required number of nearest neighbors. Our experimental results demonstrate that our structure facilitates the search for low and high dimensional data and achieves faster construction times.

5. REFERENCES

[1] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces," *Proc. of Int. Conf. on Very Large Data Bases*, pp. 426−435, 1997.

[2] C. Traina, A. Traina, B. Seeger, and C. Faloutsos, "Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes," *EDBT*, Vol. 1777, pp. 51−65, 2000.

[3] J. C. Traina, A. Traina, R. S. Filho, and C. Faloutsos, "How to Improve the Pruning Ability of Dynamic Metric Access Methods," *Proc. of the 11th Int. Conf. on IKM*, pp. 219−226, 2002.

[4] T. Skopal, J. Pokorn, and V. Snasel, "PM-Tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases," *Advances in Databases and Information Systems (ADBIS)*, 2004.

[5] M. R. Vieira, C. T. Jr., F. J. T. Chino, and A. J. M. Traina, "Dbm-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data," *Brazilian Symposium on Databases (SBBD)*, pp. 163−177, 2004.

[6] G. Navarro, "Searching in Metric Spaces by Spatial Approximation," *The VLDB Journal*, Vol. 11, pp. 28−46, 2002.

[7] J. Jaromczyk and G. Toussaint, "Relative Neighborhood Graphs and Their Relatives," *Proc. of IEEE*, Vol. 80, pp. 1502−1517, 1992.

[8] O. U. Florez and S. Lim, "HRG: A Graph Structure for Fast Similarity Search in Metric Spaces," *Proc. of the 19th Int. Conf. on Database and Expert Systems Applications*, pp. 73−81, 2008.