Group Reverse Nearest Neighbor Search using Modified Skip Graph

[1]Upinder Kaur, Research Scholar, Dept. of Computer Science and Applications, KUK.
[2]Dr. Pushpa Rani Suri, Professor, Dept. of Computer Science and Applications, KUK.

Abstract: The reverse nearest neighbor search is used for spatial queries. The reverse nearest neighbor search, the object in high dimensional space has a certain region where all objects inside the region will think of query object as their nearest neighbor. The existing methods for reverse nearest neighbor search are limited to the single query point, which is inefficient for the high dimensional spatial databases etc. Therefore, in this paper we proposed a group reverse nearest neighbor search which can find multiple query objects in a specific region. In this paper we proposed method for group reverse nearest neighbor queries using modified skip graph.

Index terms*: group reverse nearest neighbor search, spatial query, skip graph.*

*****

1. INTRODUCTION:

Reverse nearest neighbor (RNN) is a query technology and able to find the sets of database points that have the query point as the nearest neighbor. RNN has received lot of attention from the database research community because of its broad application like marketing, decision support, resource allocation and data mining, especially digital maps to process geographical information. Many of our activities today are done by the aid of spatial data. The most common one is for navigation where people can easily look for directions to specific places. Not only for navigation, we can also find some nearest objects in our surroundings. There is a rapid development in spatial databases area.

Various researches on the RNN search [1–11] have been conducted over the last decade. In [1–6], methods for finding the reverse nearest neighbor from a snapshot of a dataset are devised. The authors in [7–11] propose methods for the continuous nearest neighbor search which find the reverse nearest neighbor and continuously update the result as the objects change their locations. All the researches use only the spatial distance for measuring the distance between objects. However, in many real-life location based services, we are given more information than just the physical locations of objects. Recent services such as Google Maps, Facebook and Groupon provide users with rating scores of products. Such information is valuable in defining a new distance measure in the RNN search. For example, when choosing a premium-grade steakhouse for dinner, we generally consider not only the spatial proximities of steakhouses, but also the quality of the steakhouse based on items such as the food, atmosphere, price and service. Therefore, in order to choose the better steakhouse, the spatial proximity and the quality of the steakhouse are necessary to be comprehensively considered. In such a case, the traditional distance measure based solely on the spatial distance cannot be used. Consequently, the distance measure based on both the spatial proximity and the quality of the item is more useful and realistic than the traditional measure.

We can find all the objects that consider the query object as their nearest neighbour [1]. Another example of RNN Query is when we are looking for a set of customers who thinks of a specific supermarket as the nearest one based on the customer’s location. The RNN is different from Nearest Neighbour (kNN) Queries where the query object looks for its own nearest neighbours [5], the important thing in RNN is that whether the neighbouring objects consider the query point as their nearest neighbour [2], [6], [7]. It does not matter for the query object to know its neighbours. Like in real life, it is not substantial for a shop to look for its competitor but it is more important to understand which other shops see it as competitor. However most of the works done on this approach can only be implemented to a single query object. In real life we will encounter a situation where we need to find the RNN of more than one object. Imagine a case where we need to build a refugee camp in a war. We absolutely want to have the safest location possible, where it is closer to the military soldier locations rather than the enemies. The soldiers are usually spread in several locations. If we only rely on a single military soldier location, the area retrieved will not be safe enough to build the camp. Hence in this paper we try to solve this kind of problem by proposing the Group Reverse kNN Queries, where a region will be retrieved through multiple query objects. The region itself will always have the minimum sum of distance to the query objects altogether, which this means that when we locate a new object inside the region, the new object will always consider all of the query objects as its nearest neighbor.

As a significant role of Group Reverse kNN Query is in finding the most efficient region based on multiple query points, the region produced can help us in decision making process in our daily life like placing a new object in a strategical area. People want to find a gas station such that the total cost for visiting the gas station and filling the gas is minimum based on their locations. Therefore, the marketing targets of a gas station are promising buyers for which visiting the gas station is more economical than visiting other gas stations. The method for our problem can find such promising buyers by considering the spatial distance

In this paper, we proposed method to process the group reverse nearest neighbor queries using modified skip graph.

2. Related work

The concept of group reverse nearest neighbor query is in fact similar to the RNN of multiple query objects. The RNN results need some modification to solve the problem of Group RNN. In the past years, various approaches to solve the RNN Queries had been proposed. The initial one is by using point-to-point approach [3], [13-15], where we will gather all the possible points (objects) in the space that will consider the query as their RNN. The most common technique for this approach is to determine the Euclidean distance between every two objects. So in this case we need to check the distance between the query object with another object, one by one, in order to get the query's RNN Voronoi Diagram has been widely used in spatial query processing [16]–[19], specifically to solve the RNN Queries [17]. In Voronoi Diagram, a plane is divided into some regions and each of the regions has a generator point where any objects inside the corresponding region think of the generator point as their closest generator point compare to the points in other regions [18], [19]. The algorithms used in the Voronoi Diagram varies depending on the consideration of the Euclidean distance between objects or the real road network [17], [20]. The nature of query objects in Group Reverse kNN Queries is pretty similar to the Order-*k* Voronoi Diagram. All the query objects need to be close to each other in order to create a region. The Group Reverse kNN region itself is similar to a single Voronoi cell in Order-*k* Voronoi Diagram. A dual of Voronoi Diagram is the Delaunay Triangulation [18], [21]. The triangulation itself is formed by the generator points of Voronoi Diagram [19]. The algorithm for Delaunay Triangulation uses divide-and-conquer approach, which runs in $O(_N \log_N)$ time [22], [23]. In relation to the Group Reverse kNN Queries, the Delaunay Triangulation can be used to validate the location of the query objects. The Delaunay Triangle is applied to all of the objects in the plane, both the query objects and the non-query objects. The existing approaches in RNN may not be suitable for solving RNN of multiple query object. Like the concept used in Influence Zone, it is similar to Group Reverse kNN Query. We will have a region that when an object is placed inside this region, the object will consider the query as its nearest neighbour. However the main difference here is that Influence
Zone only focuses on a single query, while Group Reverse kNN problem is on multiple query objects. Therefore some modifications to the method are needed so that it is applicable to multiple query objects in Group Reverse kNN Query. Moreover, a study on processing multiple query objects in spatial database has previously been done by [24], [25] through the Group Nearest Neighbour (Group-KNN) method. It is a variant of kNN where a number of selected query points will choose a target object that is considered as their nearest neighbour [25]–[28]. There are numerous ways to process Group *k*NN Queries, some of them are Multiple Query Method (MQM), Single Point Method (SPM), Minimum Bounding Box Method (MBM), and Group closest pairs method [27]. Even though Group-KNN processes multiple query objects, the main focus in Group-KNN is to find a single target object, which is the centroid of the query objects [24]. This is mainly different from the purpose of Group Reverse kNN Query where we are trying to find a region, where all objects inside this region will always think of the query objects as the nearest neighbours. Moreover, the Group-KNN does not consider the non-query points when finding the centroid, while the Group Reverse kNN Query will need to consider the non-query points in order to form a region through perpendicular bisector of each query points with all other non-query points.

3. Problem definition:

The concept of RNN is to have a set of objects that considers a query object as the nearest neighbour. In this case, the query object of RNN Query has an "influence" to the other objects. The set of objects that are the result of RNN are known as the influence set of the corresponding query [2].

In the initial technique for solving RNN, point-to-point approach is applied [3], [13 -15]. This kind of method is fairly expensive because we need to verify every object one by one whether it is considered as nearest from the query object. So instead of using point-to-point approach, region approach is used [8], [9]. Since a query has influence to others, then there exists a region where any object located inside this region will be considered as the RNN objects of the query, which is why the concept of Influence Zone is presented. Supposed that we have a set of objects *P* in the *R*2 space, *P* = *{p*1, *p*2, ..., *pn}* (TABLE I). We choose one of the objects, say *p*1, as the query object. Based on the concept of Influence Zone, if an object *p* is located outside the region of object *p*1, *p* will not consider *p*1 as its nearest neighbour, *p* will consider other object instead. However when we have more than one object as the query, this approach cannot be used. The Influence Zone of a single object is different from a set of objects. In the Figure 1, we have object *p*1, *p*2, *p*3, *q*1, *q*2, and *q*3 in the space and each of their Influence Zone is also presented. We choose *q*1, *q*2, and *q*3 as the query objects, so in this case we will have *k* = 3. The rest of the non-query objects are assumed as the competitor objects. We want to have a region where any objects located inside this region will think of all the query objects as the nearest. Since we have three query objects, then when an object located anywhere inside the region performs 3NN search, object *q*1, *q*2, and *q*3 will always be the answer. If we use the Influence Zone of a single object, the region created will not always have all the query objects as the nearest. Assume that we put a new object *p* anywhere inside the region of *q*1, as can be seen in Figure 2 and Figure 3. The new object *p* will obviously think of *q*1 as the nearest. But when we do 3NN search, it will not necessarily think of *q*1, *q*2, and *q*3 as the nearest. In Figure 3, it is still as what we want since *p* considers *q*1, *q*2, and *q*3 as the nearest. But it is different from Figure 4. *p* in Figure 4 thinks of *p*1 and *p*3 as the nearest instead of *q*2 and *q*3.

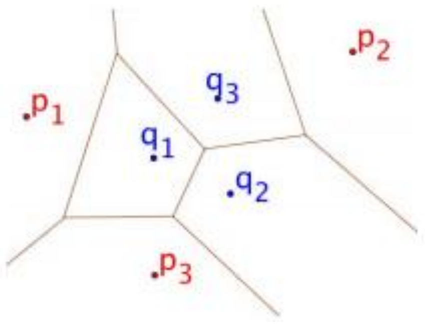

Figure 1: Influence Zone applied to each object

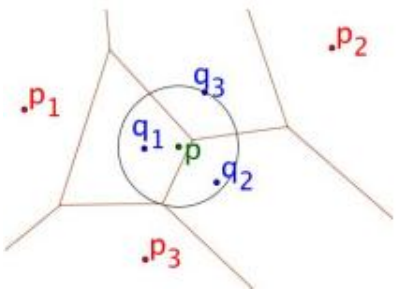

Figure 2 : *p* considers *q*1, *q*2, *q*3 as its 3NN

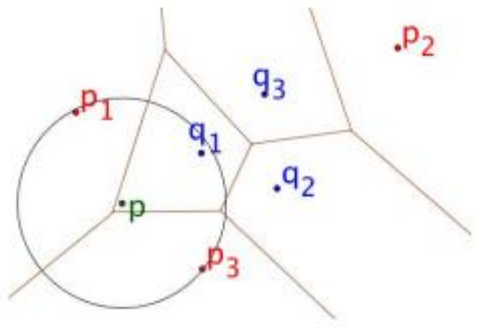

Figure 3: *p* considers *q*1, *p*1, *p*3 as its 3NN Because of the problem mentioned before, we try to find a potential region of *k* number of query objects, where all the *k* query objects will be considered as the nearest neighbor when we do *k*NN search. The region itself is not the convex hull of all the query objects. Having convex hull as the region is not accurate since there are cases where an object will also think of the competitor objects as the nearest even though it is located inside the convex hull. There also a case where the region may actually be larger than the convex hull, which makes the convex hull method to be inefficient. As can be seen in Figure 4, the Group Reverse kNN region of *q*1, *q*2, and *q*3 is actually larger compare to when we create the convex hull (shown by the black triangle) of the query objects. There are actually more region can be covered by the query, compare to when the convex hull method is applied. Hence in this paper we propose the algorithm to create an optimal region for this problem. In the Group Reverse kNN Queries, we will have *k* number of query objects as the input, and the output will be a region. a number of predefined query objects *Q* = *{q*1, *q*2, ..., *qk}*, where $qk \subseteq P$ (TABLE I), are needed in order to construct a region. The predefined query objects need to be validated by their closeness so that a region can be formed. The produced region is expected to have the shortest sum distance to the query objects and any object inside the region is believed to have all of the query points as the nearest objects. Based on our research, the region that we are looking for is created based on the influence zone of each query object. Each of the query object will have its own region, where the combination of all query object's regions will intersect and form the Group Reverse kNN Region. Figure 5 shows this example. We find the region of each query object, which are *q*1 (yellow), *q*2 (green), and *q*3 (blue). All of the regions will have an intersection, which as long as we place an object inside this intersecting region, object *q*1, *q*2, and *q*3 will always be the nearest objects compare to the competitor objects. The nature of query points in Group Reverse kNN Queries is similar to the Order-*k* Voronoi Diagram. All the query objects need to be close to each other in order to create a region [24]. Furthermore, based on the experiment

in creating region in Group Reverse kNN Queries, the region itself is similar to the Voronoi cell in Order-*k* Voronoi Diagram. The region created by *k* number of query objects in Group Reverse kNN is actually a Voronoi cell in Higher Order Voronoi Diagram. Hence, the Group Reverse kNN Queries is actually a type of query where we can create a single Voronoi cell of Order-*k* Voronoi Diagram.

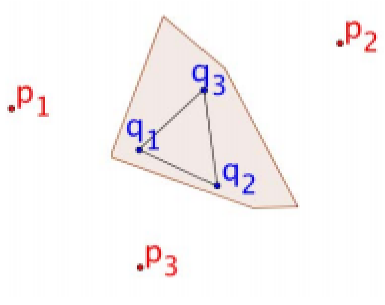

Figure 4: Region of multiple query objects

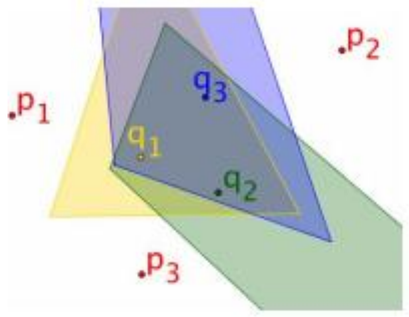

Figure 5: Region of each query object

TABLE I: List of Notations

Notation	Definition
$P \in R^2$, $P = \{p_1, p_2, ..., p_n\}$	Set of objects/points in the space
$Q = \{q_1, q_2, ..., q_k\}$, where $q_k \subseteq P$	Set of query objects/points
$C = (P - Q)$	Set of competitor points
$CH(P)$	Convex hull of set P
$CH(Q)$	Convex hull of set Q
$DT(P)$	Delaunay triangulation of set P
e_x	The edge that connects point q_i and q_j
$Tree(Q)$	The set of edges e_x
$Bis(q_k : c_n)$	Perpendicular bisector line between q_k and c_n
IZ_q	Influence zone of q

4. Proposed method

Skip Graphs extend Skip Lists for distributed environments by adding redundant connectivity and multiple handles into the data structure. It is equivalent to a collection of up to Skip Lists (where n is the number of elements in the lowest level) with each element participating in exactly one list at each level and some of the lower levels shared across many Skip Lists. The increased connectivity provides greater fault tolerance and avoids hot-spots as any element could be located using any one of the top-level elements of the different Skip Lists. The insertion,deletion and searching algorithms in modified skip graph is same used in standard skip graph.

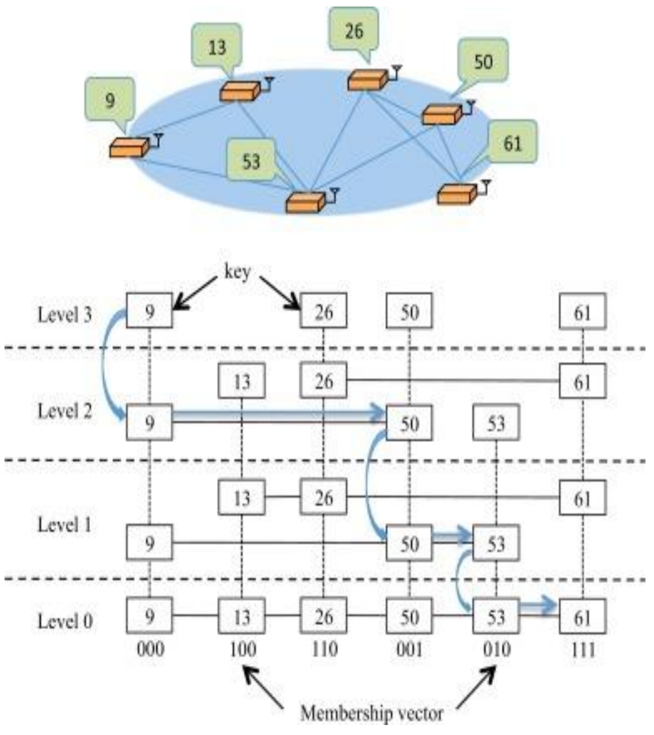

Figure 6: Example of Skip Graph

4.1 Index strategy

Regions: A region stands for a node in the region tree tree. Each region is a hyper-rectangle in the search space. Intermediate regions generate two child regions when split along one of the dimensions. Leaf regions are associated with physical machines that are responsible for managing that portion of the index.

Split history: of a region represents the path in the region tree from the root to the region. It is a list of tuples (dim $_{split}$, pos $_{split}$)with each element specifying the dimension and position of a split.

Region code: is a string of 0s and 1s that represents how a region is generated by the splitting process. When we split a region into two pieces, the region code for the left child (which is the region with smaller coordinates in the splitting dimension) is generated by appending “0” to the end of the region code of the parent region. The right child’s region code is obtained by appending “1” to the parent’s region code. We represent the code in a binary fraction format, with a decimal point to the left of the most significant bit. The ordering imposed by the region codes corresponds to the in-order (or "left-to right") traversal of the region tree. When combined with the split history, the region code completely describes the coordinates of a region.

4.2 Reverse k Nearest Neighbor Query Algorithm

Reverse k nearest neighbor search is similar to a spherical range query, except that the radius of the query ball is not prespecified but is instead determined dynamically during the search. Our reverse nearest neighbor algorithm is therefore based on the k nearest neighbor search algorithm, which is enhanced with a demand-driven process for determining which regions to query. As the first step, we route the reverse k nearest neighbor search to the node a owning the region that contains the query point, then executes a local query to determine an initial candidate set of nearest neighbors, denoted by S_{nn}. The maximum distance from q to points in S_{nn} is the initial value for the RNN query radius R_{nn} If the size of is S_{nn} less than k, then R_{nn} is set to a value that covers the entire d dimensional space. A also maintains a priority queue, Q_{search} of regions to be searched, ordered by their minimum distances to the query point. The initial contents of Q_{search} is determined by traversing a's partial tree view and finding all regions that intersect with the current query ball(q, R_{nn}).The algorithm queries the regions in Q_{search} increasing order of distance. In each step, a extracts the minimum distance region, R, from the queue and sends a query messag(q, R_{nn},R) towards region R. When this message is received by a node, which could have more detailed information about R, the search region is refined based on the node's partial tree view and forwarded towards the sub-region of R that is closest to 0 . When the search region is eventually refined to a leaf region and received by the corresponding node, a local query is performed to determine points contained in the region that are closer than the current KNN distance estimate R_{nn} . The results are reported back to a along with the newly discovered sub-regions that intersect with the query ball. A then updates S_{nn} and R_{nn} , and inserts sub-regions found during the step into $Q_{search.}$ A repeats the query step until there are no regions left in Q_{search} within distance R_{nn}. Algorithm provides the formal description

Algorithm 1: Reverse kNearset neighbor

```
A does a local nearest neighbor search initializes $S_{nn}$ and $R_{nn}$ based on the results
A traverse its partial tree view and initializes $Q_{search}$ with all the regions R, such that mindist(R,q)<=$R_{nn}$
While ($Q_{search}$ not empty) do
R←extract_min($Q_{search}$)
Forward query(q, $R_{nn}$,R) to R
Receive result ($Set_P$,$Set_R$)
Insert regions in $Set_R$ into $Q_{search}$
Update Snn and Rnn with SetP
Prune Qsearch with new Rnn
Return Snn
Upon node B receiving the query (q, $R_{nn}$,R):
If(local_region (B)==R) then
$Set_P$← reverse nearest points within $R_{nn}$
$Set_R$←leaf regions inside R from local tree view
Reply to A the results( $Set_P$.$Set_R$)
Else find the leaf $R_l$ in the local tree view with minimum mindist(q,R)
Forwardquery (q, $R_{nn}$,R) to $R_l$
```

4.3 Group Reverse kNN Queries

The main idea is to create the group reverse kNN region similar to kNN region. However we need to make sure that the query provided can be used to create the region . We do not want to waste the computation and resources in the region creation process only to find out that the region itself cannot be retrieved. So a validation will be conducted prior to creating the region. The algorithm to generate the Group Reverse kNN Queries is broken down into two main parts: (i) Query Validation and (ii) Region Generation.

Query Validation

Since we are dealing with multiple query objects in the Group Reverse kNN Query, the region may not be retrieved all the time. It actually depends on the closeness of the query objects. There is a possible case where the query points given does not produce any results, such as when the competitor objects are intermingling in between the query objects. Because of this, there is no specific region produced by Group Reverse kNN since no region serves the shortest distance to all the query objects. Hence we need to validate the query given by user. There is no specific region produced by Group Reverse kNN. There is no intersecting region of all the query objects' regions due to $p1$ and $p2$ interruption. $p1$ and $p2$ has their own influential zone as well which this makes the query objects to be not having a Group Reverse kNN region. Therefore the query objects needs to be close to each other in order to form a region. There are two levels of validating the query. First we use Delaunay Triangulation to see the connectivity of the objects, and the second one is by using the Smallest Enclosing Circle technique for all the query objects to check the closeness and clustering of the objects. For the first validation, the Delaunay Triangulation DT is applied to all of the objects in the plane P , both the query points and the competitor points. So in this case we have $DT(P)$. Through $DT(P)$, we need to see the edge $Edge(qi, qj)$ between the query objects in the triangulationwhether they are directly connected to each other or not. *Edge*(*qi*, *qj*) means that *qi* and *qj* are segment in *DT*(*P*) for *i* = *j*.

B. Region Creation

Once we have a valid query, then a region can be generated. In this step, perpendicular bisector will need to be applied for each query object. The perpendicular bisector *Bis*(*qk* : *cn*) will be between each query and other competitor points in the space. However, there is a case where we do not need to apply *Bis*(*qk* : *cn*) to all of the query points. We can create a convexhull of set *Q* (*CH*(*Q*)). All query objects that are located in the convex hull *CH*(*Q*) edges are going to be processed. The other query points inside the *CH*(*Q*) do not need to be processed since the region of those query points will always be inside the region of query points located in the *CH*(*Q*) edges. This method can save up some processing time as we do not need to process the whole query objects, but we rather have the most influential ones. So if *qx* lies inside *CH*(*Q*), then its influence zone *IZ qx* will always cover the intersection of all influence zone of query points that are located in *CH*(*Q*) edges. The example can be seen in

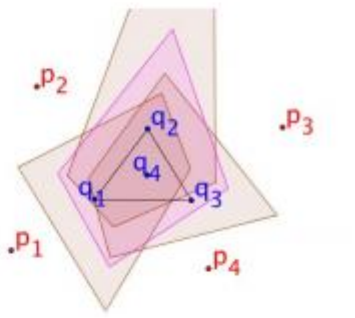

Figure 8. The query objects are *q*1, *q*2, *q*3, and *q*4. *q*4 is located inside the *CH*(*Q*) of the query objects.

```
Algorithm 2: Group Reverse kNN Algo
Data: set of query objects Q; set of objects in the space P
Result: Group Reverse kNN region
Begin
    C=P-Q;
    Boolean qV=QueryValidation(Q,P)
    If qV is True then
        CreateRegion(Q,C)
    End
End
```

Thus, *q*4 is the *innerQ* and then *q*1, *q*2, *q*3 are the *outerQ*. The distance of *q*4 to *p*1 is greater than the distance from *q*1 to *p*1. So when we apply perpendicular bisector to *q*4 with *p*1 and *q*1 with *p*1, the length of *q*4 to the *Bis*(*q*4 : *p*1) is greater than the length from *q*1 to *Bis*(*q*1 : *p*1), which this makes the region *q*4 to be always inside the regions of objects located in *CH*(*Q*) edges. The brown shaded area in the figure are the regions of *q*1, *q*2, *q*3, while the purple area is the region of *q*4. Figure 8: Convex hull of *q*1, *q*2, *q*3, *q*4 with regions of each query objects As we have known which query objects to be processed, we also need to consider the competitor objects. Not all the competitor objects in the space will be used in creating the region of each query. Thus in this case we can prune those unused competitor objects. We will only choose the competitor objects that have direct impact in the region creation process.

```
Algorithm 3: Query Validation
Data: set of query objects Q;set of objects in the space P
Result: query valid or invalid(true or false)
Begin
    DT(P)←apply Delaunay Triangulation to P
        If Tree(Q)<l ot Tree(Q)>l then
            Return false;
        End
    SEC(Q)←apply smallest Enclosing Circle to Q;
        If no competitor inside SEC(Q) then
            Return true
        End
    true
End
```

The pruning process in creating influence zone has already been solved by Adhinugraha *et al.* [29] through the usage of Contact Zone. The idea of Contact Zone is to find potential peers that have impact to the region generation of the query, which are identified by the peers location inside the Contact Zone. It will eliminate all unneeded peers so that not all the points in the space will be processed. This method is efficient in saving up time and resources. Hence we will apply this existing method to our algorithm. After we have pruned the unused competitor objects, we can now apply the perpendicular bisector *Bis*(*qk* : *cn*) between each query with all unpruned competitors. This will give us a region of the corresponding query object. Each of the query will have their own region. All of those regions will have an intersection, which this created the Group Reverse kNN Region. The intersection area itself can be obtained through polygon clipping algorithm [32], [33]

```
Algorithm 4: Create Region
Data: set of query objects Q; set of non-query object C
Result: Group Reverse kNN region
CH(Q)←create convex hull for Q based on the Delaunay Triangualation lines;
    For each $q_k$ in CH(Q) edges do
        Apply Contact Zone;
        Apply Bis($q_k$:$c_n$);
    End
  Get the intersection region
End
```

5. EVALUATION

In this section, some evaluations are conducted in order to determine the efficiency of the algorithm proposed. There are. several cases evaluated for each method, which are aiming to examine the efficiency of both the validation and region generation process. We mainly focusing on the closeness of the query objects in the query validation process. So the variation of number of query objects is important in the evaluation. We generate 100 to 1000 objects as the sample, and then randomly choose 2 to 5 objects as the query. Several cases are applied to the query objects, both in average and extreme cases. We chose the query to be located in the middle of the competitors, where the query is surrounded by the competitors. We also chose the query in the corners where the competitors are gathering in a specific side of the query objects, not surrounding them. In evaluating the Smallest Enclosing Circle method, we analyse the effect of competitor objects inside the circle. An increasing number of competitor objects are added randomly inside the circle. The location of the competitor objects has effects to the region, hence we try to apply as many plot as possible on each number of competitors in order to get the percentage of un-retrieved region. Figure 9 shows the effect of competitor objects inside the smallest enclosing circle in the validation process and when average c0ase is applied. It shows that the chance of region would not be retrieved is very high when the competitors are inside, especially when the number of both the query objects and competitors increase. There are chances the region can be retrieved but only when the number of query and competitor objects are very low.

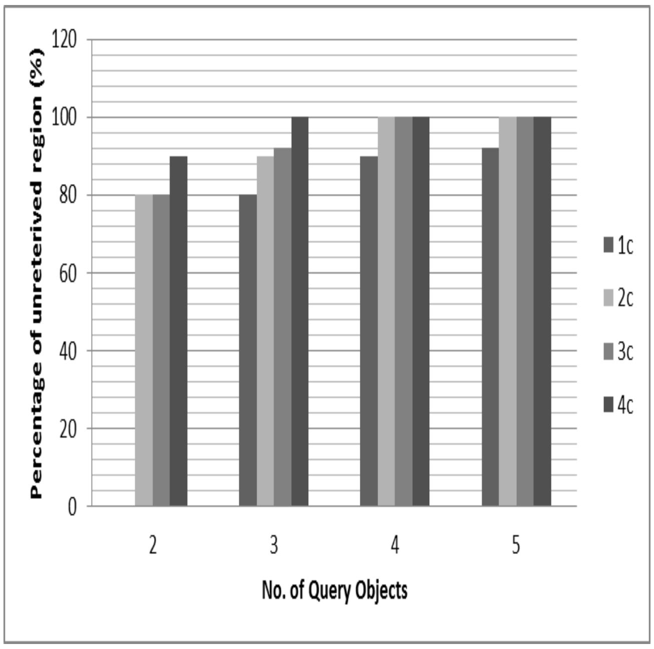

Figure 9:Effect of Competitors inside the circle on average cases

However when extreme case is applied on the Group Reverse kNN queries, the chance of region to be retrieved is slightly bigger than in the average case. This is due to the case where the competitor objects are gathered in a specific side of the query objects. The graph in Figure 10 still shows a considerably rise of percentage of un-retrieved region whe the number of query and competitor objects increase. Hence based on this evaluation, the chance of region to be created is pretty low when the competitor lies inside the Smallest Enclosing Circle. It is not suggested to process the query with un-retrieved region as we will only waste the resources. Another evaluation is in the region creation process where we analyse the number of competitor objects that are being pruned. The competitor objects need to be pruned in order to safe up some computation since not all of the competitors will be used to create the region. In our evaluation, we apply the Contact

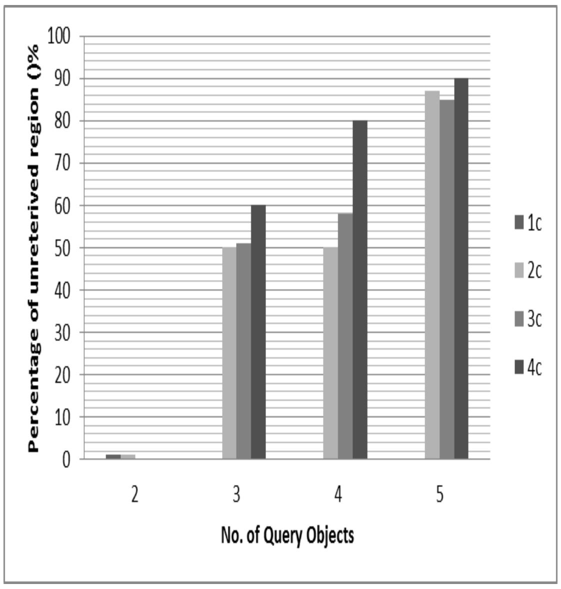

Figure 10: Effects of competitors inside the circle on extreme case

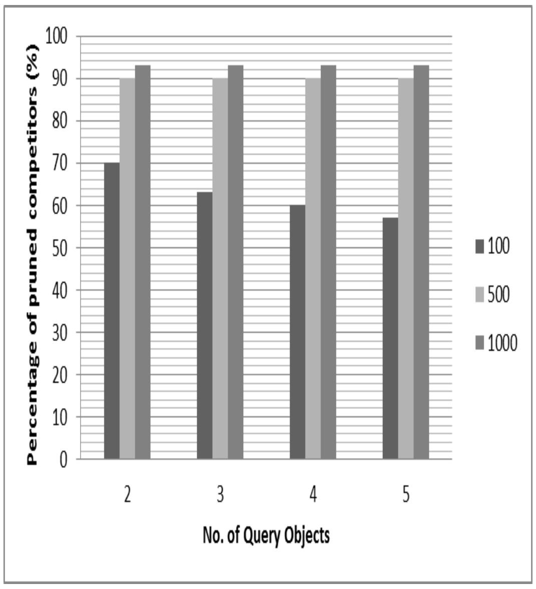

Figure 11:Competitiors pruned before creating

Zone method to each of the query object and count the number of competitor objects that are being pruned. Figure 11 shows the result of the pruned competitors for 100 to 1000 objects in the space. As can be seen in the graph, the number of pruned competitors are decreasing as the number of query objects increase. This is as expected since the more the number of query, the more the competitors that affecting the region creation. However, the percentage of pruned competitors shows that it is very efficient since most of the unused competitors are being pruned, especially on the high density case since it can even eliminate more than 80% of the competitor objects.

6. CONCLUSION

The Group Reverse kNN Query is a type of spatial query that processes a set of query objects in order to find a potential region where all objects inside this region will think of the query objects as the nearest neighbour. This spatial query has significant roles in our daily life, such as in business and urban planning. In this paper we proposed the algorithm to process Group Reverse kNN Queries. The algorithm is broken down into two main parts, which consists of Query Validation and Region Creation process. We evaluated the efficiency of both of the processes. The evaluation on the validation method shows that the chance of region to be created is pretty low when the competitor lies inside the Smallest Enclosing Circle. Another more accurate approach on the validation may be needed as our future work. The evaluation on the region creation shows that the method is efficient in pruning the unused competitor objects. However another future work can be conducted in maximising the pruning of the competitor objects since current method needs to apply the Contact Zone to each of the query objects one by one.

References:

[1] F. Korn, S. Muthukrishnan, Influence sets based on reverse nearest neighbor queries, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD'00, ACM, New York, NY, USA, 2000, pp. 201–212.

[2] C. Yang, K.-I. Lin, An index structure for efficient reverse nearest neighbor queries, in: Proceedings of the 17th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 2001, pp. 485–492. URL ⟨http://dl.acm.org/citation.cfm?id=645484.656392⟩.

[3] K.-I. Lin, M. Nolen, C. Yang, Applying bulk insertion techniques for dynamic reverse nearest neighbor problems, in: 2003 Proceedings of Seventh International Database Engineering and Applications Symposium, 2003, pp. 290–297.

[4] I. Stanoi, D. Agrawal, A.E. Abbadi, Reverse nearest neighbor queries for dynamic databases, in: In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 44–53.

[5] Y. Tao, D. Papadias, X. Lian, Reverse knn search in arbitrary dimensionality, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases, vol. 30, VLDB'04, VLDB Endowment, 2004, pp. 744–755.

[6] W. Wu, F. Yang, C.-Y. Chan, K.-L. Tan, Finch: evaluating reverse k-nearest-neighbor queries on location data, Proc. VLDB Endow. 1 (1) (2008) 1056–1067.

[7] R. Benetis, C.S. Jensen, G. Karciauskas, S. Saltenis, Nearest neighbor and reverse nearest neighbor queries

for moving objects, in: Proceedings of the 2002 International Symposium on Database Engineering & Applications, IDEAS'02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 44–53.

[8] T. Xia, D. Zhang, Continuous reverse nearest neighbor monitoring, in: Proceedings of the 22nd International Conference on Data Engineering, ICDE'06, IEEE Computer Society, Washington, DC, USA, 2006, p. 77.

[9] J. Kang, M. Mokbel, S. Shekhar, T. Xia, D. Zhang, Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors, in: IEEE 23rd International Conference on Data Engineering, 2007, ICDE 2007, 2007, pp. 806–815.

[10] W. Wu, F. Yang, C.Y. Chan, K.-L. Tan, Continuous reverse k-nearest-neighbor monitoring, in: Proceedings of the Ninth International Conference on Mobile Data Management, MDM'08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 132–139.

[11] M.A. Cheema, X. Lin, Y. Zhang, W. Wang, W. Zhang, Lazy updates: an efficient technique to continuously monitoring reverse knn, Proc. VLDB Endow. 2 (1) (2009) 1138–1149.

[12] J. Lu, Y. Lu, G. Cong, Reverse spatial and textual k nearest neighbor search, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD'11, ACM, New York, NY, USA, 2011, pp. 349–360.

[13] Y. Tao, M.L. Yiu, N. Mamoulis, Reverse nearest neighbor search in metric spaces, IEEE Trans. Knowl. Data Eng. 18 (9) (2006) 1239–1252.

[14] A. Singh, H. Ferhatosmanoglu, A.c. Tosun, High dimensional reverse nearest neighbor queries, in: Proceedings of the Twelfth International Conference on Information and Knowledge Management, CIKM'03, ACM, New York, NY, USA, 2003, pp. 91–98.

[15] M.L. Yiu, D. Papadias, N. Mamoulis, Y. Tao, Reverse nearest neighbors in large graphs, IEEE Trans. Knowl. Data Eng. 18 (4) (2006) 540–553.

[16] A. Vlachou, C. Doulkeridis, Y. Kotidis, K. Norvag, Reverse top-k queries, in: 2010 IEEE 26th International Conference on Data Engineering (ICDE), 2010, pp. 365–376.

[17] M. Safar, D. Ibrahimi, and D. Taniar, "Voronoi-based reverse nearest neighbor query processing on spatial networks," Multimedia Systems, vol. 15, no. 5, pp. 295–308, 2009.

[18] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, and D. G. Kendall, Definitions and Basic Properties of Voronoi Diagrams, pp. 43–112. John Wiley & Sons, Inc., 2008.

[19] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, "Voronoi diagrams," in Computational Geometry, pp. 147–171, Springer Berlin Heidelberg, 2008.

[20] H. Cho, S. J. Kwon, and T. Chung, "A safe exit algorithm for continuous nearest neighbor monitoring in road networks," Mobile Information Systems, vol. 9, no. 1, pp. 37–53, 2013.

[21] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, "Delaunay triangulations," in Computational Geometry, pp. 191–218, Springer Berlin Heidelberg, 2008.

[22] D. Lee and B. Schachter, "Two algorithms for constructing a Delaunay triangulation," International Journal of Computer & Information Sciences, vol. 9, no. 3, pp. 219–242, 1980.

[23] L. Paul Chew, "Constrained delaunay triangulations," Algorithmica, vol. 4, no. 1-4, pp. 97–108, 1989.

[24] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in Data Engineering, 2004. Proceedings. 20th International Conference on, pp. 301–312, March 2004.

[25] T. Nghiem, D. Green, and D. Taniar, "Peer-to-peer group k-nearest neighbours in mobile ad-hoc networks," in Parallel and Distributed Systems (ICPADS), 2013 International Conference on, pp. 166–173, Dec2013.

[26] M. Safar, "Group k-nearest neighbors queries in spatial network databases," Journal of Geographical Systems, vol. 10, pp. 407–416, 12 2008. Copyright - Springer-Verlag 2008; Last updated - 2014-08-30.

[27] D. Taniar and W. Rahayu, "A taxonomy for nearest neighbour queries in spatial databases," Journal of Computer and System Sciences, vol. 79, no. 7, pp. 1017 – 1039, 2013.

[28] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," ACM Trans. Database Syst., vol. 30, pp. 529–576, June 2005.

[29] K. M. Adhinugraha, D. Taniar, and M. Indrawan, "Finding reverse nearest neighbors by region," Concurrency and Computation: Practice and Experience, vol. 26, no. 5, pp. 1142–1156, 2014.

[30] P. K. Agarwal, D. Eppstein, and J. Matousek, "Dynamic half-space reporting, geometric optimization, and minimum spanning trees," in Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on, pp. 80–89, Oct 1992.

[31] A. Efrat, M. Sharir, and A. Ziv, "Computing the smallest k-enclosing circle and related problems," Computational Geometry, vol. 4, no. 3, pp. 119 – 136, 1994.

[32] I. E. Sutherland and G. W. Hodgman, "Reentrant polygon clipping," Commun. ACM, vol. 17, pp. 32–42, Jan. 1974.

[33] S. Feng and X. Du, "A polygon clipping algorithm based on series coding technique," in Future Computer and Communication (ICFCC), 2010 2nd International Conference on, vol. 1, pp. V1–373–V1–377, May 2010.