The Nucleus **52**, *No. 2 (2015) 45-49*

www.thenucleuspak.org.pk

The Nucleus

ISSN 0029-5698 (Print)
ISSN 2306-6539 (Online)

Optimized k-Nearest Neighbor Search with Range Query

M. Rehman and T. Ahmad

Department of Computer Science and Engineering, University of Engineering and Technology, Lahore, Pakistan

madeeha.rehman5@gmail.com; Tauqir_ahmad@uet.edu.pk

ARTICLE INFO

Article history :
Received : 09 March 2015
Revised : 27 April 2015
Accepted : 30 April, 2015

Keywords :
Range Query
k-NN,
Geospatial Object
Density Based Range Query

ABSTRACT

K-Nearest Neighbor search is used extensively in fields like computer vision, DNA specification, object recognition and many more. Online map applications are also used tremendously due to the advances in Geographic Information System (GIS) data. These geospatial objects can be retrieved by using spatial range query. In our proposed work we find the nearest neighbors across any query point q, using an over estimated value of k. The Range Query calculated area is used to estimate the number of k neighbors. If the initial value of k doesn't reveal all the nearest neighbors inside R, the value of k is multiplied with a constant factor epsilon. In this way all the nearest neighbors inside R are retrieved with in 1 or 2 iterations. The computational complexity of the proposed algorithm turns out to be O(n), compared to the complexity of Density Based Range Query algorithm i.e., O log(n^2 + n). This makes our proposed algorithm a more optimized solution.

1. Introduction

Geographic Information System (GIS) is computer software that helps visualize, store and manipulate all types of spatial or geographical data. It helps us find locations like real estate site selection; route selection etc. A general issue in GIS is the Nearest Neighbor search. Nearest Neighbor search is used to find the closest (or similar) points to a query point q in a given Range [1]. In a nutshell, the problem is to find k- nearest neighbors (k-NN) in a given Range, where k is the number of nearest neighbors to find [2]. The space in which we find the number of k- neighbors known as Range Query is a rectangular window of finite area.

A Density Based (DB) Range Query Algorithm given by [3] finds nearest neighbors to a query point q, using Density estimation technique. This algorithm estimates the density of objects around the query point, and use this estimated value as the initial k value. The estimation of objects around the query location is done by global estimation method. In Global estimation technique, the algorithm finds total number of objects in the geospatial database (the GIS database in which all the location data is stored) and calculates the area of the minimum bounding rectangle (MBR) that surrounds all objects. The value of k is calculated through the formula given in equation (1).

$$k = N/A_{global} * \pi . R^2q \qquad (1)$$

Where N is the total number of objects in the geospatial database, A_{global} is the area of MBR, R is the area of the Range Query and q is the query location.

If the resultant k-NN search does not retrieve all the nearest neighbors, then another approach local estimation is used using the k value from equation (1).

$$k' = k/A_{global} * \pi . R^2q \qquad (2)$$

The local estimation is repeated as many times unless all the nearest neighbors in a Range query are retrieved. It takes up to 3 to 4 iterations to completely find all the k-NN inside the Range. This method increases the overall complexity of the DB Range Query algorithm i.e., O log (n^2 + n) and therefore cannot be used with geospatial databases where the database size is large [3].

In this research, we have proposed a k-NN Search Algorithm using Range Query. This algorithm estimates the number of neighbors around a query point according to the area of the Range query. It provides the complete coverage of all the geospatial objects inside the Range Query, with in 1 or 2 iterations of the algorithm. The complexity of proposed algorithm is calculated to be O(n). The solution to this problem is helpful in major areas like online maps, location findings and route selections [4].

2. Problem Statement

Given a database of spatial objects say 'P', we have to find only those spatial objects that fall in the nearest location of our query 'q'. The problem statement is formally defined as: To find all objects 'P' in a given region 'R' that are nearest to query 'q'.

[*] Corresponding author

The distance function is used to calculate the shortest path between the query ‘q’, and any point p_i. This function is known as the Euclidean distance or the Manhattan distance and is used to calculate the distance in k-Nearest Neighbor Query [5]. Formally we can define the Euclidean distance as: the length of a straight line between two points in space [6]. Mathematically the Euclidean distance can be calculated by the formula given in equation (3) [7]:

$$d= \sqrt{\sum_{i=1}(P_{1i} - P_{2i})^2} \quad (3)$$

Where, P_{1i} and P_{2i} are the two points between which the Euclidean distance is found.

Mathematically it can be described as given a set of point’s p_1, p_2, p_3, p_4… We find the nearest neighbor to each point according to a query, using the distance function d.

This can be mathematically written as:

If

$$P = \{p_1, p_2, p_3, p_4\} \quad (4)$$

Then

$$q = \{qp_1, qp_2, qp_3, qp_4\} \quad (5)$$

Using the value of P from equation (4) in equation (5)

$$q = q\ \{p_1, p_2, p_3, p_4\ldots\} \quad (6)$$

$$q = q\{p\} \quad (7)$$

For equation (4) and (6) it is assumed for simplicity that all the points in set P fall in the nearest neighbor location of query point q.

So, the problem statement is described as: Provided a region R, find all the spatial objects in this region using the k-NN search. In such situations the user has no control and information about how big is the database and how many nearest neighbors would be located. In order to solve this issue, we propose to use the area of the Range Query to estimate the k value. In such a way, the time taken to retrieve the k-NN is reduced. The number of times the algorithm iterates is also reduced to 1 or 2 iterations. This hence provides more fast and accurate results.

3. Proposed Solution

In this paper, we have provided an algorithmic approach that will provide an optimistic and yet efficient version of Density Based Range Query algorithm discussed earlier in section 1. Our proposed work uses only the area of the query region R, to calculate the value of k, and always provide an overestimated value of k. By [3] overestimation does not increase the cost of algorithm, the number of iterations are greatly reduced and thus the results are fast.

Suppose we have a Range Query R of some finite area. We need to find all the spatial objects scattered inside the Range Query. For this purpose we need query point ‘q’. q is kept as the centre point of R, so to search the range query equally from all sides. Then, all the objects that fall inside the Range Query R are said to be the nearest neighbors of q.

We can say that:

$$R = \{O_1, O_2 \ldots O_n\} \quad (8)$$

If k = 3, the search on R would yield:

$$k\text{-NN} = \{O_1, O_3, O_5\} \quad (9)$$

Then by evaluating equation (9) on equation (8), it is found that

$$k\text{-NN} \subseteq R \quad (10)$$

Equation (10) reveals that all the objects of k-NN must be equal or belong to the objects of R i.e., k-NN objects are inside R.

k value is determined by estimating the number of nearest neighbors in the Range Query area. This estimation yields an overestimated value of nearest neighbors that has two major benefits:

i. Overestimation doesn’t incur any cost on the system [3].
ii. Yet it decreases the number of iterations the algorithm runs, thus decreasing the execution time.

Suppose the maximum distance from q to any neighbor during a 4-nn search is found, this is the radius of k-NN circle. The k-NN circle radius is compared to Range Query radius to find if all neighbors inside the query region are found. If the result is smaller, another iteration of the k-NN search is made to increase the number of k.

The proposed algorithm is given in Table 1, producing two main benefits:

i. Cost / overhead and execution time of the algorithm is reduced.
ii. Distance Browsing will produce far better results [9].

The main focus of this proposed solution is to minimize the overhead that occurs when the algorithm is iterated again and again to find the nearest neighbors. The complexity of the algorithm is calculated using the Big O notation [10]. The proposed algorithm complexity is calculated to be O(n), as there is only one loop used in this algorithm. But, the complexity of DB Range Query Algorithm is calculated to be O $\log(n^2 + n)$, as the DB Algorithm uses two loops.

Table 1: Proposed k-NN search algorithm

k-NN Search Algorithm

```
$D_r = \sqrt{x_{ul}^2 + Y_{lr}^2}$
$q = D_r / 2$;
$K = \pi . D_r$;
Knn[] = KNNSerch(q, K)
r = mxDistKnn (q, Knn[])
while r < = $D_r$ do
clear Knn[]
$\varepsilon$ = const value
Knew = K * $\varepsilon$
Knn[] = KNNSerch (q, Knew)
r = mxDistKnn (q, Knn[])
end while
result[] = Result (Knn[], $X_{ul}$, $Y_{lr}$)
return result[]
```

4. Performance Evaluation

We have implemented this proposed algorithm and the DB Range Query Algorithm in Mat lab and have run it on a laptop with core i3 and 6 GBs of Ram. Time is a major factor while testing this algorithm. The correct time that a computer uses to run a specific piece of code is hard to find in a multitasking system. Therefore we have measured the time a number of times and then have used the average value. We have used a synthetic dataset with uniform distribution of points [11]. In a uniform distribution, the objects are distributed uniformly over a given space (x, y). On the same uniform dataset, we have implemented DB Range Query Algorithm. The time found is very large as is shown in Figure 1.

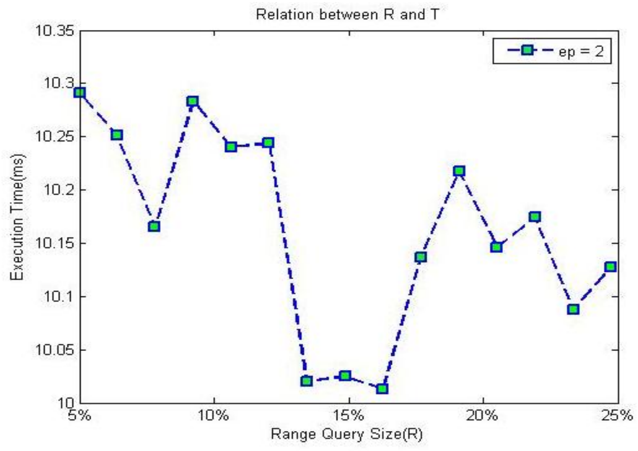

Fig. 1: DB range query algorithm, relation between range query size and execution time

The trend of the algorithm in Figure 1 is quite large; the time is always above 1 sec. We have taken the time in milliseconds and the size of Range Query in percentages as shown in figure 1. We depict how the time increases if we increase the size of the Range Query. The initial execution time of the DB Algorithm takes up to 10.3 milliseconds, on subsequent increases of the Range Query size the time decreases, but the average time taken is always greater than 10 milliseconds. Even the best case in the depiction takes time greater than 10 milliseconds, shown where the Range Query size is 15% and the time is slightly greater than 1 second. This is the major drawback of DB Range Query Algorithm.

All experiments are conducted by differing the range query size, and analyzing how the values of k nearest neighbor and time change according to it. The results are particular, the values of k nearest neighbors are increasing as we increase the size of the Range query, and this is obvious. We have changed the size of the range query from 1% to 30%. Thus the relation between Range query size R and number of k neighbor are found to be directly proportional.

4.1 *Statistical Analysis*

In Statistical analysis, we find how the size of Range query affects the number of k nearest neighbors. As the algorithm uses an overestimated k value, the Range query algorithm finds all the nearest neighbors within 1 or 2 iteration and as the size of range query increases, k value increases proportionally. The value also shows presentable change when we change the value of the constant factor epsilon ε. This is depicted in Figure 2.

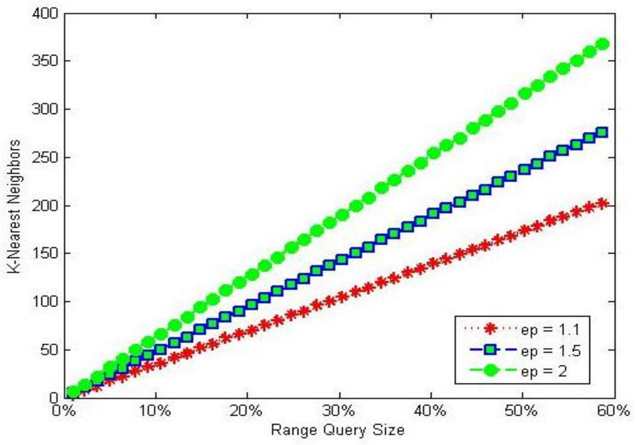

Fig. 2: Relation between range query size and k-nearest neighbor

As we see in Figure 2, the numbers of k nearest neighbor's increases proportionally on increasing the size of range query that can be found by equation (11).

$$k = \pi * R \tag{11}$$

R is radius of the Range query. And every next interval of the k value increases in a similar fashion, which can be calculated by equation 13. The size of Range query and k value is increased with small intervals.

$$R = R + 1.39 \tag{12}$$

$$k = k + 5 \tag{13}$$

By using the value of R by (12) in equation (13)

$$K = \pi (R + 1.39) \quad (14)$$

By using the value of k from equation (13) in equation (11)

$$R = (k + 5) / \pi \quad (15)$$

Thus, we have 2 formulas for calculating the next interval values of k-Nearest Neighbor and size of Range Query. Now we will see how time effects the calculations of k nearest neighbors. Whether we increase the range query size or the number of k nearest neighbors found is increased, the efficiency of the algorithm is not affected. That is the time it take to run the algorithm shows a similar fashion as in Figure 3. We have experimented the time by two ways. By increasing the value of Range Query and observing the time. Secondly, observing the time curve on different values of k.

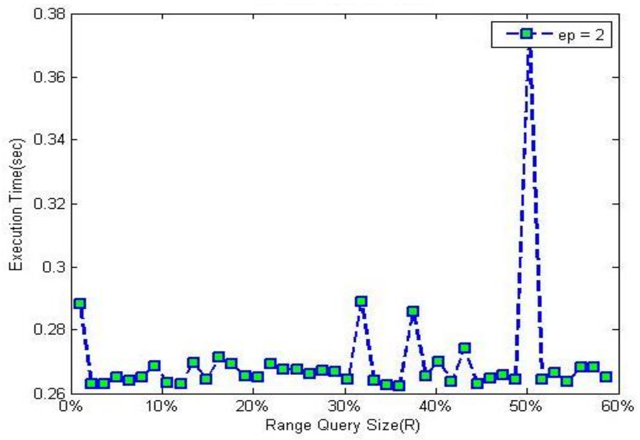

Figure 3: Relation between time (seconds) and range query size (%age)

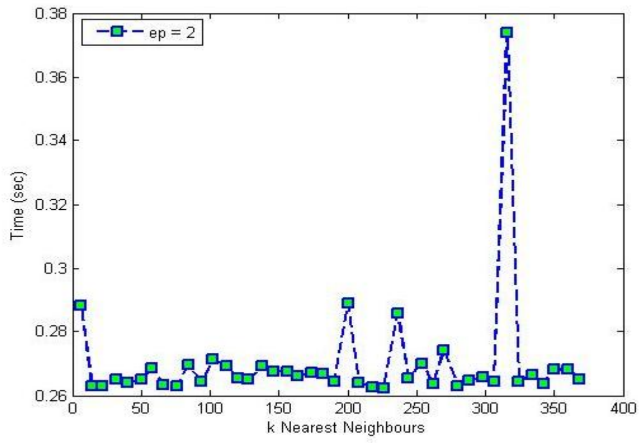

Figure 4: Relation between time and values of k-NN

We have increased the Range Query size from 1% to 60%, and studied the effect of execution time on Range query size and k nearest neighbors found. We have trained a large set of data points (neighbor locations) to find out, what would happen if query size and the number of nearest neighbors increase and to see the effects on distance browsing [9]. And it is found that, whether the size of k nearest neighbors increase or the Range Query size is increased, the time it takes to find the k nearest neighbor remains same.

For simplicity, we have only shown the results where the constant value is epsilon $\varepsilon = 2$. The trends for $\varepsilon = 1.5$ and $\varepsilon = 1.1$ shows similar fashion. We can see here the time curve is similar in Figure 3 and Figure 4. Thus it is proved that the relation between k and R is directly proportional. From Figure 2 the graph line of k-NN increases as the R size increases. This relationship is shown in equation (16).

$$k \propto R \quad (16)$$

The time curve depicts that the worst case occurs once in the life time of the algorithm, at 50% Range Query Size it takes up to 0.38 seconds. All other values the execution time remains inside 0.26 seconds, without any effect on whether we increase the size of the Range or the database size increases. In another case shown in figure 3 and figure 4, when the Range Query size is 30% and the number of k– neighbors founds are above 200. The algorithm takes 0.285 seconds to reveal all the neighbors. The data points scatter is kept large for all the calculations performed. So to check how the algorithm behaves if the size of the number of spatial objects increases, in terms of the execution time taken to retrieve them. The resultant accuracy of our proposed algorithm is also greater than DB Range Query Algorithm. As, our proposed algorithm finds more nearest neighbors in less time, it thus yields the accurate results in lesser number of iterations. While the DB range Query algorithm takes more time, and more number of iterations to find the accurate result.

We can say that the number of k nearest neighbors increases as we increase the size of the Range Query and the execution time remains under 0.38 seconds. The algorithm finds the complete number of neighbors in one or two iterations by overestimating the value of k. As we see the number of k neighbors does not affect the efficiency of algorithm, proves that this can work best with distance browsing.

5. Results and Discussion

The comparison between the Density Based Range Query Algorithm and Proposed Algorithm is shown in Table 2. For both algorithms, same sizes of the Range query are used, and both algorithms are executed equal number of times. We have increased the Range Query Size from 3% to 21% and have calculated the effects of both the algorithms. Our proposed approach finds more neighbors in less time. But the DB Range Query Algorithm produces very small k-Nearest Neighbors.

Table 2 shows different k values and the corresponding execution time at different Range Query sizes. For all executions of the Density Based Algorithm, the execution time is more than 1 second. While, the proposed Algorithm gives overestimated values of k, and the execution time does not exceed 0.3 seconds on average. The maximum number of k-Neighbors founded by the DB Range Query Algorithm is k = 60. On similar number of iterations, our proposed algorithm gives k = 138 and the execution time is dramatically lower, as shown in Table 2. The DB Range Query Algorithm takes a maximum time of 1.3 seconds to find k-Nearest Neighbors.

Table 2: Comparison of execution time between the density based range query algorithm and the proposed algorithm

Range Query Size	Density Based Range Query Algorithm		Proposed Algorithm	
	Number of k-Nearest Neighbors	Execution Time (seconds)	Number of k-Nearest Neighbors	Execution Time (seconds)
3%	2	1.047310	22	0.263060
5%	4	1.175698	32	0.265016
7%	8	1.193790	50	0.265244
9%	10	1.049320	58	0.268675
11%	14	1.051839	66	0.263270
13%	22	1.040362	84	0.269638
15%	28	1.043013	94	0.264292
17%	38	1.036382	112	0.269141
19%	44	1.051083	120	0.265594
21%	60	1.268208	138	0.269423

6. Conclusion

In this research, we have designed an algorithm to search k number of nearest neighbors according to the query point ‘q’ in a Query region ‘R’. The algorithm finds the area of the region and then calculates the number of k neighbors in that region. The proposed algorithm finds maximum number of k neighbors in minimal time, and does not use complex calculations. The major inspiration for this algorithm is taken from Density Based Range Query algorithm. In DB Range Query algorithm, global and local estimation techniques are used for area calculation, which increases the overhead of execution.

References

[1] Y. Tao, J. Zhang, D. Papadias and N. Mamoulis, "An Efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces", Journal of IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 10, pp. 1169-1184, 2004.

[2] K.A. Hawick, P.D. Coddington and H.A. James, "Distributed frameworks and parallel algorithms for processing large scale geographic data," Journal of Parallel Computing - Special issue: High Performance Computing with Geographical Data, vol. 29, no. 10, pp. 1297-1333, 2003.

[3] W.D. Bae, S. Alkobaisi, S.H. Kim, S. Narayanappa and C. Shahabi, "Web data retrieval: Solving spatial range queries using k-Nearest neighbor searches", Geo Informatica , vol. 13, no. 4, pp. 483-514, 2009.

[4] G. Beskales, M.A. Soliman and I.F. Ilyas, "Efficient search for the top-k probable nearest neighbors in uncerain databases", Proceedings of the VLDB Endowment, vol. 1, no. 1, pp. 326-339, August 2008.

[5] B. Aydin, "Parallel algorithms on nearest neighbor search", Survey Paper for Parallel Algorithms - Georgia State University, April 2014.

[6] K. Katu and T. Hosino, "Solving k-nearest neighbor problem on multiple graphics processor", Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 769-773, 15 July, 2010.

[7] T. W. P. Series, "Euclidean Distance, raw, normalized and double-scaled coefficients", Advanced Projects R&D, September 2005. [Online]. Available: http://www.pbarrett.net/techpapers/euclid.pdf.

[8] D. Liu, E.P. Lim and W.K Ng, "Efficient k nearest neighbor queries on remote spatial databases using range estimation", Procceedings of 14th International Conference on Scientific and Statistical Database Management, pp.121-130, 26 July, 2002.

[9] G.R. Hjaltason and H. Samet, "Distance browsing in spatial databases", ACM Transactions on Database Systems, vol. 24, no. 2, pp. 265-318, 1999.

[10] P. Danziger, "Big O Notation", 2015. [Online]. Available: http://www.wikipedia.org/w/wiki.phtml?title=Big_O_notation.

[11] V. Garcia, E. Debreuve and M. Barlaud, “Fast k nearest neighbor search using GPU”, IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, Anchorage, AK, 23-28, pp. 1-6, June 2008.