Journal of Computer Science 10 (6): 1045-1050, 2014
ISSN: 1549-3636

doi:10.3844/jcssp.2014.1045.1050 Published Online 10 (6) 2014 (http://www.thescipub.com/jcs.toc)

FINDING A RESIDENCE WITH ALL FACILITIES USING NEAREST NEIGHBOR SEARCH

Padmapriya, K. and Dr. S. Sridhar

Department of Computer Science and Engineering, Sathyabama University, Chennai, India

Received 2013-11-12; Revised 2013-11-14; Accepted 2014-02-01

ABSTRACT

Nearest neighbor search is one of the most widely-used techniques and its applications including mobile communication, Geographic information systems, bioinformatics, computer vision and marketing. For example, four friends want to rent an apartment which should be nearer to their working places. Our paper discussed about the problems on finding the most appropriate location among a set of available places. The problem is defined as a top-k query which gives output of k points from a set of available places P along with the conveniences. We proposed algorithms based on R-trees to answer the query exactly. The efficiency of our proposed algorithms is verified through various experiments and found that it is better than existing algorithms use large scale real datasets.

Keywords: Nearest Neighbor Search, R-Trees, Local Priority Queue, Global Priority Queue

1. INTRODUCTION

Nowadays Optimal locations problems got the great focus on research (Corral *et al*., 2004; Wong *et al*., 2009). In this study, we discuss about a new problem of finding the place with all facilities like a restaurant, super market and a bus stop nearby. The person may opt for different choices according to his interests. Furthermore he may give higher priority to certain choice when compared with other choices. Hence we use monotonic function to set priorities.

We tried to solve this problem in Euclidean distance like the other existing works on the facility location problems (Du *et al*., 2005; Zhang *et al*., 2006) by fulfilling the criteria of facilities asked by the user. Already many algorithms have been proposed for all nearest neighbors problem (Chen and Patel, 2007; Emrich *et al*., 2010) and aggregate nearest neighbor problem (Li *et al*., 2005; Mouratidis *et al*., 2005; Papadias *et al*., 2004). But in these papers, only one facility has been taken into account but ours will work out with multiple facilities with certain preferences.

We put forward two algorithms, first one making the facilities in separate R-trees and traverses it to result into top-k queries. The second algorithm ranking the facilities in a R-tree and finds out the most suitable settings. Both algorithms are experimented carefully with many non-trivial optimizations results.

By keeping the existing algorithm as our baseline algorithm, we keep on reporting the users about the best sites based upon their facility criteria. Suppose a user is satisfied with first n results, he can terminate the algorithm.

In this study we proposed 2 algorithms for finding out the most appropriate site among a set of possible sites. Our algorithm is used to reduce the running time and I/O cost. We experimented our algorithm with real datasets and results show better performance than the existing algorithm.

2. PROBLEM SETUP

We defined two spatial datasets S and F. Every point f in F is numbered a type n. Let total number of types is |T|. The Euclidean distance d(s,f) measures the distance between s and f. The cost is calculated as cs of s by using the Equation (1):

$$cs = f(d(s, NN(s, Fn)),....d(s, NN(s, F|T|)) \quad (1)$$

Corresponding Author: Padmapriya, K., Department of Computer Science and Engineering, Sathyabama University, Chennai, India

where, NN(s,Fn) is the nearest location of i from s and f() is a function which takes input |T| as parameters and results into a single value. This paper aims to locate different k points from S with minimal cost among all the points in S.

Then we construct the R-tree RS for locations in S and the second R-tree RFn for each type of facilities in F. Therefore |T|+1 R-trees are there. Then we start examining these two R-trees starting from their roots. We use Local Priority Queue (LPQ) and Global Priority Queue (GPQ) from previous works for determining the access order of nodes in R-trees.

LPQ: LPQ is assigned to each node m in RS and maps its entries in RFn. The entries p in the priority queue has mindist and maxdist, represent minimum distance computed from m's MBR to p's MBR and the maximum distance computed from m's MBR to p's MBR. The priority queue is ordered by the ascending order of mindist values and maxdist is used for pruning the unpromising nodes. Apart from these two values, minmindist represents the minimum value in the mindist values defined from its entries and minmaxdist represents minimum of maxdist values defined in its entries.

GPQ: A min heap is used for pruning the LPQs, ordered by ascending order of minmindist.minmaxdist.

GPQ and LPQ are used to find the most promising nodes with the lower bound of smallest distance. To avoid duplicate nodes in RS, we allow only one LPQ for each node in RS.

The nearest neighbor algorithms iterate all over the categories in F. As in (Chen and Patel, 2007; Shin *et al*., 2000), each iteration for a type n, started from the roots of RS and RFn and expand the it in a bi-directional manner. If end level is attained in RS and RFn, then the nearest neighbors are returned. After finishing expansion of all types in F, it returns k points to the top-k queries with minimum accessibility cost. The disadvantage of this algorithm is that RS will be traversed only |T| times not |T|+1 times. Hence we will not get the results till all the types are processed.

3. ALGORITHMS

We use existing algorithms as our baseline algorithm and enhancing it with two more new algorithms for fulfilling many facilities defined by the user.

3.1. Isolated Tree Method

This method is used for choosing the indexes and traversing the trees in a parallel manner like the all nearest neighbor algorithm. We use the datastructures of Global Priority Queue (GPQ) and Local Priority Queue(LPQ) in our isolated tree method. Since we tried to expand all entries of R-trees in a parallel manner, each entry m in RS has its own |T| LPQs and call it as LPQm. So we can estimate the accessibility cost of the points in m. We calculate the lower bound as Equation (2):

$$\text{LowBoundc} \sum_{n=1}^{|T|} \text{LPQm}[n].\min\text{indist} \quad (2)$$

where, LPQm[n] maintains the entries in RFn. We push LPQs to GPQ from the smallest value of LowBoundc to highest value of LowBoundc. The top-k results are stored in a min-heap H and the results obtained temporarily with k^{th} minimum cost for accessibility is stored in H[k]. We evaluate the newest LPQ groups' LowBoundc accompanied with the cost of H[k]'s accessibility. If it is smaller, then the entries are made on new LPQ groups.

Algorithm1: Isolated Tree (RS, RF)

```
For each point s in S do
        Cs←NULL;
H←InitializeTempResults;
GPQ←NULL;
For n =1 to |T| do
        m← RS.root;
        p←RFn.root;
        LPQm[n] NULL;
        LPQm[n].minmindist← ∞;
        LPQm[n].minmaxdist ← ∞;
        IsoTreePushAndUpdate(LPQm[n],p);
GPQ.push(LPQm);
While GPQ ≠ NULL do
        LPQm←GPQ.pop();
        IsoTreeExpansion(LPQm, GPQ);
```

Algorithm 1 explains isolated tree algorithm. Here we have initiated the min heap H by means of selecting some k points which are temporary results obtained at the beginning. Those points are calculated with accessibility costs of all nearest neighbors. Iteration starts from RS's root and RFn's root and expanding its nodes in a bi-directional manner. Initially, the formed LPQ is hold by the root of RS. Then RFn's root is pushed into the priority queues. There after the group of LPQ is inserted into GPQ. We keep on selecting a group of LPQ from GPQ and expanding its nodes in both RS and RFn.

Algorithm 2: IsoTreePushAndUpdate(LPQm[n], p)

```
if mindist(m,p) < LPQm[n].minmaxdist then
        LPQm[n].push(p);
        LPQm[n].minmindist ←
min(LPQm[n].minmindist,mindist(m,p));
        LPQm[n].minmaxdist←
min(LPQm[n].minmaxdist,maxdist(m,p));
        LPQm.LowBoundc←
                ∑_{n=1}^{|T|} LPQm[n].min indist
```

Algorithm 3 explains the expansion algorithm. Having a node m in RS, the entries stored in m's LPQ are deleted, according to mindist. We find out the nearest neighbor by summing the distance with the cost of accessibility and updating the temporary results till the point is reached in both the R-trees RS and RFn. Or else the deleted entry of p is paired with the children of m to form a new LPQ group. Especially, we expand m for each type and new group is created for each of its children m. The priority queue p's children are pushed into the LPQ of m□. We evaluate the nodes'mindist of m□ and minmaxdist of ml. If the mindist is lesser than m's minmaxdist, then we pushed that node into m□'s LPQ. Once the entry is made, the LPQ's minmindist and minmaxdist values are updated. At last we will check LowBoundc of newly created LPQ group prior to pushing it into GPQ. If the cost of its accessibility is lesser than LowBoundc of LPQ group deleted from GPQ, then the temporary results are declared as final results.

Algorithm 3: IsoTreeExpansion(LPQm, GPQ)

```
If ml is a point then
        For n=1 to |T| do
                whileLPQm[n] # NULL do
                        p←LPQm[n].pop();
if p is appointed then
                        cm←cm + d(m,p);
                        if m's NNs of all categories
are originated and cm <H[k].cost, then
                                H.add(m,cm);
                                Return
                        else
                                For each p□Î p do
        IsoTreePushAndUpdate (LPQm[n],p□);
        if LPQm.LowBoundc < H[k].cost then
                GPQ.push(LPQm);
Else
        For each m□∈ m do
                LPQm[n]□ ← NULL;
                LPQm[n]□.minmindist ← ∞;
                LPQm[n]□.minmaxdist ← ∞;
        For n=1 to |T| do
                While LPQm[n] # NULL do
                        p ← LPQm[n].pop();
                        if p is a point then
                                for each p□∈ p do

        IsoTreePushAndUpdate(LPQm[n],p);
else
                        For each p□∈ p do
                                For each ml∈ml do

        IsoTreePushAndUpdate(LPQm[n],p);
        For each m□∈ m do
                If LPQm.LowBoundc < H[k].cost then
                        GPQ.push(LPQm);
```

4. EXPERIMENTAL RESULTS

We have implemented our algorithm and by using the data from http://en.wikipedia.org/wiki/Highways_of_Tamil_Nadu.

We separated it into |T| types and if it is not specified, the default |T| is 20. A random type is assigned for each point in the dataset. To generate the possible sites S, we select 15% of the original dataset points and modify the x, y coordinates randomly in the range of [-10,10]. We evaluate the expanded interior nodes and expanded leaf nodes in R-trees along with the processing time which does not include construction of R-tree indexes.

We have executed our algorithm with different values of objects (k) and categories of points(|T|). **Figure 1-3** show the performance of our algorithm based on different k. Here we have noticed that running time of our algorithm slightly increases when the k values become larger. The result proves that our algorithm is 1.5 times faster than the existing ANN algorithm. It has been achieved by the expansion of internal node and leaf node very efficiently.

Then we examined the concert of our algorithm based on different categories of points |T| and its results have been shown in **Fig. 4-6**. In this experiment, we have taken 100 as default k value. As mentioned in the

figures, the node expansion and the processing time are grown linearly when |T| is increasing.

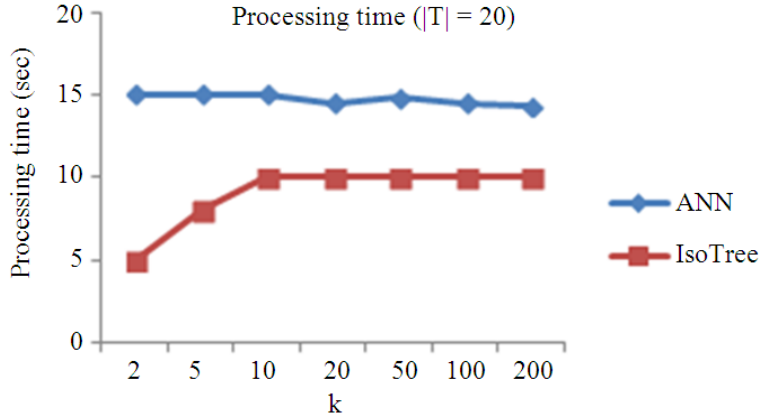

Fig. 1. Processing time w.r.t. |T|

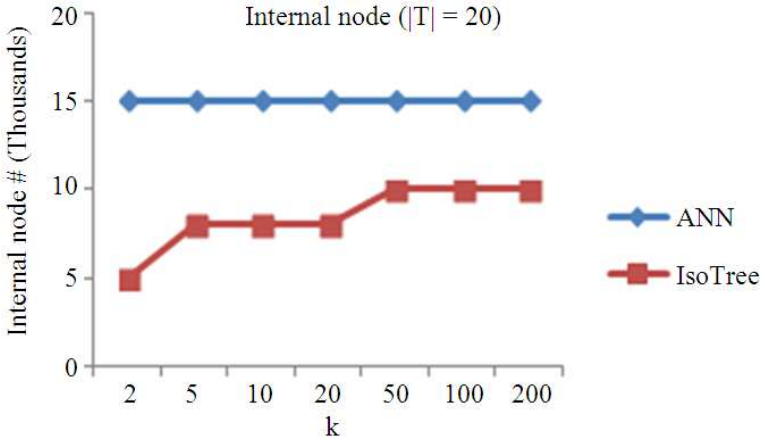

Fig. 2. Internal node expansion w.r.t. |T|

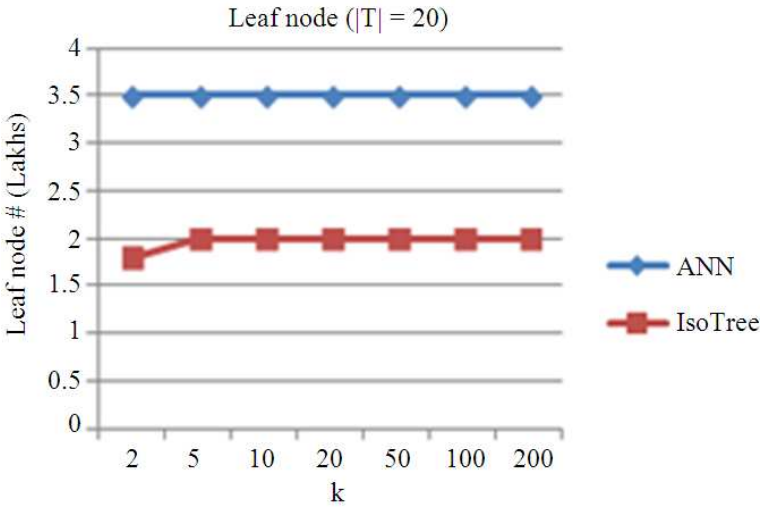

Fig. 3. Leaf node expansion w.r.t. |T|

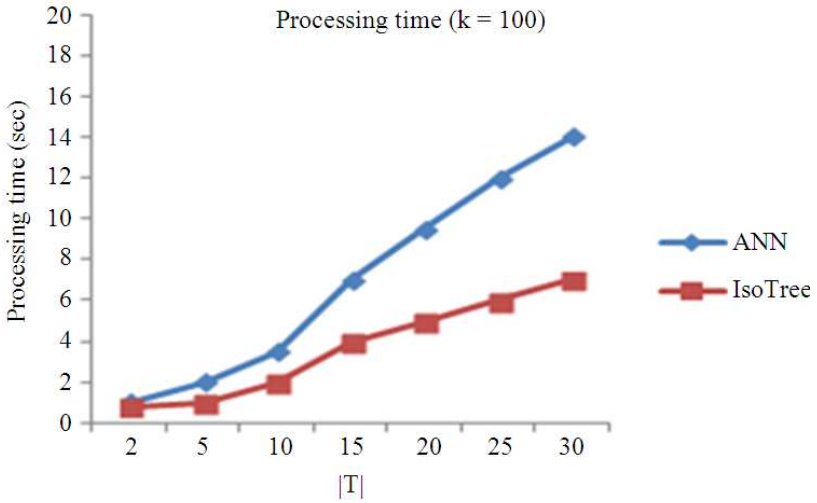

Fig. 4. Processing time w.r.t. k

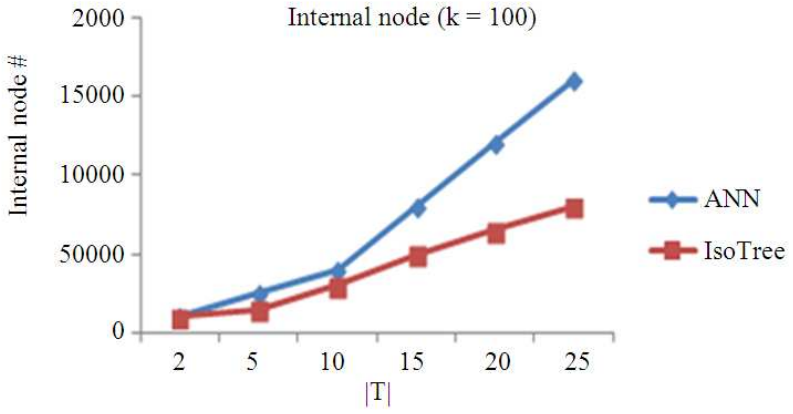

Fig. 5. Internal node expansion w.r.t. k

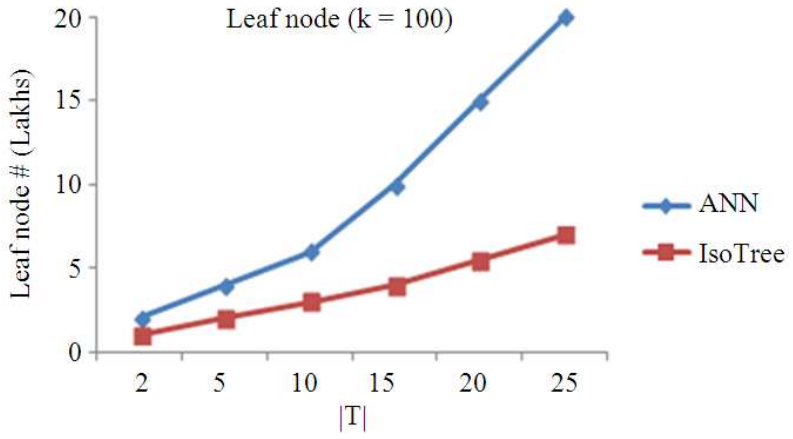

Fig. 6. Leaf node expansion w.r.t. k

5. CONCLUSION

Here we studied the problem of finding k best sites which are nearer to different types of facilities. We computed the accessibility with the help of sum of distances to nearest neighbors using Euclidean distances. Our proposed algorithm finds the top-k locations efficiently by improving runtime performance. The experimental results prove that our algorithm outperforms the baseline algorithm.

6. REFERENCES

Chen, Y. and J.M. Patel, 2007. Efficient evaluation of all-nearest-neighbor queries. Proceeding of the 23rd International Conference on Data Engineering, Apr. 15-20, IEEE Xplore Press, Istanbul, pp: 1056-1065. DOI: 10.1109/ICDE.2007.368964

Corral, A., Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos, 2004. Algorithms for processing k-closest-pair queries in spatial databases. Data Knowl. Eng., 49: 67-104. DOI: org/10.1016/j.datak.2003.08.007

Du, Y., D. Zhang and T. Xia, 2005. The optimal-location query. Proceeding of the 9th International Conference Advances in Spatial and Temporal Databases, Aug. 22-24, Springer Berlin Heidelberg, Brazil, pp: 163-180. DOI: 10.1007/11535331_10

Emrich, T., F. Graf, H.P. Kriegel, M. Schubert and M. Thoma, 2010. Optimizing all-nearest-neighbor queries with trigonometric pruning. Proceeding of the 22nd International Conference Scientific and Statistical Database Management, Jun. 30-July 2, Springer Berlin Heidelberg, Germany, pp: 501-518. DOI: 10.1007/978-3-642-13818-8_35

Li, H., H. Lu, B. Huang and Z. Huang, 2005. Two ellipse-based pruning methods for group nearest neighbor queries. Proceedings of the 13th Annual International Workshop on Geographic Information Systems, Oct. 31-Nov. 05, ACM Press, Bremen, Germany, pp: 192-199. DOI: 10.1145/1097064.1097092

Mouratidis, K., M. Hadjieleftheriou and D. Papadias, 2005. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. Proceedings of the International Conference on Management of Data, Jun. 13-17, ACM Press, Baltimore, MD, USA., pp: 634-645. DOI: 10.1145/1066157.1066230

Papadias, D., Q. Shen, Y. Tao and K. Mouratidis, 2004. Group nearest neighbor queries. Proceedings of the 20th International Conference on Data Engineering, Mar. 30-Apr. 2, IEEE Xplore Press, pp: 301-312. DOI: 10.1109/ICDE.2004.1320006

Shin, H., B. Moon and S. Lee, 2000. Adaptive multi-stage distance join processing. Proceedings of the International Conference on Management of Data, May 15-18, ACM Press, Dallas, TX, USA., pp: 343-354. DOI: 10.1145/335191.335428

Wong, R.C.W., M. T. Ozsu, P.S. Yu, AW.C. Fu and L. Liu, 2009. Efficient method for maximizing bichromatic reverse nearest neighbor. Proc. VLDB Endowment, 2: 1126-1137.

Zhang, D., Y. Du, T. Xia and Y. Tao, 2006. Progressive computation of the min-dist optimal-location query. Proceedings of the 32nd International Conference on Very large Data Bases, (DB'06), ACM Press, pp: 643-654.