

Top-K Spatial Preferences Based On Quality Features

R.JAYAMMA, Asst.Professor, Dept.of CSE,

MADHIRA INSTITUTE OF TECHNOLOGY & SCIENCES, KODAD, NALGONDA, ANDHRA PRADESH.

Abstract: A spatial preference query ranks objects based on the qualities of features in their spatial neighborhood. For example, using a real estate agency database of flats for sale, a customer may want to rank the flats with respect to the appropriateness of their location, defined after aggregating the qualities of other features (e.g., restaurants, market, hospital, railway station, etc.) within their spatial neighborhood. Such a neighborhood concept can be specified by the user via different functions. In this paper, we study an interesting type of preference queries, which select the best spatial location with respect to the quality of facilities in its spatial neighborhood. For example consider the office construction, we want gather the requirements with respect their spatial neighborhood like creation of right of way. Using our proposed work quality preferences of special data allocation can be done efficiently.

I INTRODUCTION

The recent top-k spatial preference queries ranks objects based on the spatial neighborhood quality features. Spatial database systems manage large collections of geographic entities, which apart from spatial attributes contain non-spatial information (e.g., size, type etc.). In this paper, we study an interesting type of preference queries, which select the best spatial location with respect to the quality of facilities in its spatial neighborhood.

Traditional spatial queries and joins mainly focused on distances and manipulating only spatial locations, but they ignore the importance of quality attributes. The dominance comparison is suitable for comparing two objects with respect to multiple quality attributes. In this system, we study an interesting type of spatial queries, which select the best spatial location with respect to the quality of facilities in its spatial neighborhood. Given a set D of interesting objects (e.g., candidate road locations), a

top-k spatial preference query retrieves the k objects in D with the highest scores.

Traditionally, there are two basic ways for ranking objects: 1) spatial ranking, which orders the objects according to their distance from a reference point, and 2) non spatial ranking, which orders the objects by an aggregate function on their non spatial values. The top-k spatial preference query integrates these two types of Ranking in an intuitive way. Despite the usefulness of the top-k spatial preference query, to our knowledge, it has not been studied in the past. A brute-force approach for evaluating it is to compute the scores of all objects in D and select the top-k ones. This method, however, is expected to be very expensive for large input datasets.

For example, consider road networks which consists of different features like location of road, type, name if exist etc. Based on this features, we retrieve the top-k spatial preference query retrieves the k objects i.e., roads with highest scores..The neighborhood of an object is captured by the scoring

function: 1) The range score restricts the neighborhood to a crisp region centered at p , whereas 2) the influence score relaxes the neighborhood to the whole space and assigns higher weights to locations closer to p . We presented probing algorithms for processing top- k spatial preference queries.

II RELATED WORK

Assuming that the set of interesting objects is indexed by a hierarchical spatial access method (e.g., the R-tree), we can use distance bounds while traversing the index to derive the answer in a branch and-bound fashion. Tao et al. noted that top- k queries can be modeled as (weighted) nearest neighbor queries, in the multi-dimensional space defined by the involved attribute domains, where the query point is formed by taking the maximum value of each dimension. Our techniques apply on spatial partitioning access methods and compute upper score bounds for the objects indexed by them, which are used to effectively prune the search space.

In spatial databases, ranking is often associated to nearest neighbor (NN) retrieval. Given a query location, we are often interested in retrieving the set of nearest objects to it that qualify a condition (e.g., roads, restaurants). Nevertheless, it is not always possible to use multidimensional indexes for top- k retrieval. First, such indexes usually break-down in high dimensional spaces. Second, top- k queries may involve an arbitrary set of attributes from a set of possible ones and indexes may not be available for all possible attribute combinations (i.e., they are too expensive to create and maintain). Third, information for different rankings to be combined (i.e., for different attributes) could appear in different

databases (in a distributed database scenario) and unified indexes may not exist for them.

Spatial Query Evaluation on R-trees

Given a spatial region W , a spatial range query retrieves from R the objects that intersect W . A nearest neighbor (NN) query takes as input a query object q and returns the closest object in R to q . A popular generalization is the k -NN query, which returns the k closest objects to q , given a positive integer k . NN (and k -NN) queries can be efficiently processed using the best-first (BF) algorithm, provided that R is indexed by an R-tree. A priority queue PQ which organizes Rtree entries based on the (minimum) distance of their MBRs to q is initialized with the root entries. The aggregate R-tree (aR-tree) is a variant of the R-tree, where each non-leaf entry augments an aggregate measure for some attribute value (measure) of all points in its subtree. R-trees can efficiently process main spatial query types, including spatial range queries, nearest neighbor queries, and spatial joins Feature-based Spatial Queries.

Related to top- k influential query are the optimal location queries. The goal is to find the location in space (not chosen from a specific set of roads) that minimizes an objective function. Recently, solved the problem of finding top- k sites (e.g., roads) based on their influence on feature points (e.g., junctions, individual roads). Here, assume that all feature points have the same quality. The maximum influence optimal location query finds the location (to insert to the existing set of roads) with the maximum influence, whereas the minimum distance optimal location query searches for the location that minimizes the average distance from each feature point to its nearest site. The optimal

locations for both queries are marked as white points..

III Algorithms for Spatial Preference Queries

The techniques proposed are specific to the particular query types described above and cannot be extended for the class of top-k spatial preference queries and they deal with a single feature dataset whereas our queries consider multiple feature datasets. By implementing top-k spatial preference query, we will retrieve the k points in an object dataset D (i.e., set of interesting points like junctions or roads) which has the highest score. subtree. Nevertheless, our solutions are generic and it can be adapted where the datasets are indexed by other hierarchical spatial indexes (e.g., point quad-trees).

The rationale of indexing different feature datasets by separate R-trees is that: (i) out of all possible features (e.g., road networks, hospital, market, etc.), a user queries for only few features which are required (e.g., roads), and (ii) different subsets of features are considered or used by different users. Based on the indexing schemes which are specified in related work, we implement various algorithms for processing top-k spatial preference queries which are listed below:

Probing Algorithms:

Initially, we introduce a brute-force solution that computes the score of every point p which belongs to dataset D in order to obtain the query results. Then, we propose to implement a special

group evaluation technique that computes the scores concurrently for multiple points.

Simple Probing Algorithm:

To implement this algorithm, global variables are used. They are: W_k is a min-heap for managing the top-k results and represents the top-k score so far (i.e., lowest score in W_k). The simple probing algorithm (SP) compute the score for every object point present in dataset then retrieve the query results based on the computed scores. Initially, the algorithm is started at the root node of the object tree (i.e., $N = D.root$). The procedure is recursively applied on tree nodes until a leaf node is accessed. Score is calculated at each node until it reaches the leaf node. When a leaf node is reached, the component score $\tau(e)$ is computed by executing a range search (NN search) on the feature tree F_c for range score (NN score) queries

Group probing Algorithm

Group Probing Algorithm: Due to separate score computations for each and every object in data set, SP is inefficient when large object datasets. To overcome the above drawback, we implement the group probing algorithm (GP) which is a variant of SP, that reduces overall I/O cost by computing scores of objects in the same leaf node of the R-tree concurrently.

Branch and Bound Algorithm

In GP, when a leaf node is visited, its points are first stored in a set V and then their component scores are computed concurrently at a single traversal of the F_c tree. In order to overcome the problem of SP, GP was introduced but it is expensive. GP

examines all objects in D and computes their component scores. Now, we propose an algorithm that can significantly reduce the number of objects to be examined. The key idea is to compute, for non-leaf entries e in the object tree D , an upper bound $T(e)$ (but not $\tau(e)$) of the score for any point in the subtree of e .

Now, recall that $T(e)$ which is an upper bound score for any point in e . With the component scores $T_c(e)$ known so far, we can derive a new score called $T^+(e)$, an upper bound of $T(e)$. If $T^+(e) \leq \gamma$, then the subtree of e cannot contain better results than those in W_k and it is removed from V . BB(Branch and Bound) is called with N being the root node of D . If N is a leaf node then we update the set W_k of the top- k results with concurrently computed scores for all points of N . If N is a non-leaf node, compute the scores $T(e)$ concurrently for non-leaf entries. We sort the entries in descending order of $T(e)$ then we invoke the procedure recursively on child node which was pointed by the entries presented in V , in order to obtain points with high scores. Since both W_k and γ are global variables, the value of γ is updated during recursive call of BB.

Algorithm1. Branch-and-Bound Algorithm $W_k = \text{new min-heap of size } k \text{ (initially empty)}$;

$\Gamma := 0$; k th score in W_k algorithm BB(Node N) 1. V

$:= \{e | e \in N\}$;

If N is nonleaf then

for $c := 1$ to m do

4. compute $T_c(e)$ for all $e \in V$ concurrently;

5. remove entries e in V such that $T^+(e) \leq \gamma$;

sort entries $e \in V$ in descending order of $T(e)$;

for each entry $e \in V$ such that $T^+(e) \leq \gamma$; do

read the child node N' pointed by e ;

BB(N');

else

for $c := 1$ to m do

compute $rc(e)$ for all $e \in V$ concurrently;

remove entries e in V such that $T^+(e) \leq \gamma$;

14. update W_k (and γ) by entries in V ;

Our major goal is to compute these upper bound scores with low I/O cost and the bounds not to be too loose, in order for pruning to be effective. For this, we utilize only level-1 entries (i.e., lowest level non-leaf entries) in F_c for deriving upper bound scores because: (i) there are much fewer level-1 entries than leaf entries, and (ii) high level entries in F_c cannot provide tight bounds. Algorithm can be used for this purpose after changing Line 2 to check whether child nodes of N are above the leaf level.

Optimized Branch-and-Bound Algorithm

To reduce the cost of the BB algorithm, we develop a more efficient score computation technique. Let p be an object point in Data set D . Suppose that we have traversed some paths of the feature trees on F_1, F_2, \dots, F_m . Let γ be an upper bound of the quality value for any unvisited entry (leaf or non-leaf) of the feature tree F_c .

algorithm Optimized_Group_Range(Trees

F_1, F_2, \dots, F_m , Set V , Value c , Value γ)

for $c := 1$ to m do

$H_c := \text{new max-heap (with quality score as key)}$;

insert $F_c.\text{root}$ into H_c ;

$\mu_c := 1$;

for each entry $p \in V$ do

6. $rc(p) := 0$;

7. $\alpha := 1$; . ID of the current feature tree

while $|V| > 0$ and there exists a nonempty heap H_c do

deheap an entry e from H_c ; 10..update threshold

11. if $\gamma := V$; mindist e pthen

12. continue at Line 8;

13. for each $p \in V$ do . Prune unqualified points
remove p from V ;
read the child node CN pointed to by e ;
for each entry e' of CN do
if CN is a non-leaf node then insert e' into H_c ;

Here, ϵ refers to the distance threshold of the range score, and γ represents the best score found so far. For each feature tree F_c , we employ a maxheap H_c to traverse the entries of F_c in descending order of their quality values. The pseudo code for computing the scores of objects efficiently from the feature trees F_1, F_2, \dots, F_m . The set V contains objects whose scores need to be computed. The root of F_c is first inserted into H_c . The variable maintains the upper bound quality of entries in the tree that will be visited. We then initialize each component score $rc(p)$ of every object $p \in V$ to 0.

IV PERFORMANCE

Our algorithms can directly be applied for such extended preference queries. A simple optimization applicable for these queries is to prune non-leaf entries at feature trees, if they are closer than γ_c , but cannot improve, due to the distance of their MBRs to the examined objects. Another extension of our queries include preferences also on the data objects. For instance, assume that we are looking for flats that are cheap, big, and close to vegetarian restaurants.

There are several ways to process such queries. BB can be optimized to prioritize the examination of data in D , based on preferences on the object attributes and prune subtrees that cannot lead to better results than the ones currently found. First, our algorithms can be used for this purpose if the object tree is also an aR-tree, where the non-spatial

attributes (e.g., type, size) are aggregated accordingly. Another method is to search primarily on the nonspatial preferences, with the use of a top-k algorithm and probe the feature sets for the spatial preference component, as long as the current top-k results can be improved. In NN score computation, we weight based on a maximum acceptable distance of a nearest neighbor. Thus, data objects with very close nearest neighbor of moderate quality may be preferable to objects with a far nearest neighbor, even if it has a higher quality.

V CONCLUSION

Initially, we implemented a baseline algorithm Simple probing which computes the scores of every object by performing spatial queries on feature datasets. But SP reduces the number of component score computations for the objects which is optimized by an incremental computation technique. Next, we presented a variant of SP i.e., group probing algorithm which reduces I/O cost by computing scores of objects in the same leaf node concurrently. Later based on GP, we developed algorithm Branch and Bound, which prunes non-leaf entries in the object tree that cannot lead to better results. For efficient results, we optimized branch and bound algorithm to reduce the cost of the BB algorithm. Spatial database systems manage large collections of geographic entities, which apart from spatial attributes contain non spatial information. In this paper, we studied top-k spatial preference queries, which provide a novel type of ranking for spatial objects based on qualities of features in their neighborhood. In our proposed technique we have to develop some Upper Bound methods for the objects,

they are used to proving the search process in the Spatial data process.

VI REFERENCES

- [1] M.L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis, "Top-k Spatial Preference Queries," Proc. IEEE Int'l Conf. Data Eng. (ICDE), 2007.
- [2] N. Bruno, L. Gravano, and A. Marian, "Evaluating Top-k Queries over Web-Accessible Databases," Proc. IEEE Int'l Conf. Data Eng. (ICDE), 2002.
- [3] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD, 1984.
- [4] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," ACM Trans. Database Systems, vol. 24, no. 2, pp. 265- 318, 1999.
- [5] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," Proc. Int'l Conf. Very Large Data Bases (VLDB), 1998.
- [6] K.S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'Nearest Neighbor' Meaningful?" Proc. Seventh Int'l Conf. Database Theory (ICDT), 1999.
- [7] C.S. Jensen, J. Kolar, T.B. Pedersen, and I. Timko, "Nearest Neighbor Queries in Road Networks," Proc. ACM Int'l Workshop Geographic Information Systems, 2003.
- [8] N. Jing, Y.W. Huang, and E.A. Rundensteiner, "Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation," IEEE Trans. Knowledge and Data Eng., vol. 10, no. 3, pp. 409- 432, 1998.

[9] I. F. Ilyas, W. G. Aref, and A. Elmagarmid. Supporting Top-k Join Queries in Relational Databases. In VLDB, 2003.

[12] N. Mamoulis, K. H. Cheng, M. L. Yiu, and D. W. Cheung. Efficient Aggregation of Ranked Inputs. In ICDE, 2006.

About Author

I am R.Jayamma working as a Assistant Professor CSE department in MITS Engineering College. I am having 10 years of experience in various subjects in Computer Science Engineering. My interest in developing and research in data mining, cloud computing, networking etc.

