International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 4, June - 2012

An Efficient Algorithm for processing Top-k Spatial Preference Queries

S Rao chintalapudi
Asst.Professor
CMR Technical Campus

katikireddy srinivas
Associate Professor
B.V.C Engineering College

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 formally define spatial preference queries and propose appropriate indexing techniques and search algorithms for them. Extensive evaluation of our methods on both real and synthetic data reveals that an optimized branch-and-bound solution is efficient and robust with respect to different parameters.

Index Terms—*Query processing, spatial preference query, spatial databases*.

1. INTRODUCTION

Spatial database systems manage large collections of geographic entities, which apart from spatial attributes contain non-spatial information (e.g., name, size, type, price, 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. Given a set D of interesting objects (e.g., candidate locations), a top-k spatial preference query retrieves the k objects in D with the highest scores. The score of an object is defined by the quality of features (e.g., facilities or services) in its spatial neighborhood. As a motivating example, consider a real estate agency office that holds a database with available flats for sale. Here "feature" refers to a class of objects in a spatial map such as specific facilities or services. A customer may want to rank the contents of this database with respect to the quality of their locations, quantified by aggregating non-spatial characteristics of other features (e.g., restaurants, super market, hospital, railway station, etc.) in the spatial neighborhood of the flat (defined by a spatial range around it). Quality may be subjective and query-parametric. For example, the user (e.g., a tourist) wishes to find a hotel p that is close to a railway station and a high-quality restaurant. Fig. 1a illustrates the locations of an object dataset D (hotels) in white, and two feature data sets: the set F1 (restaurants) in gray, and the set F2 (railway stations) in black. For the ease of discussion, the qualities are normalized to values in [0, 1].

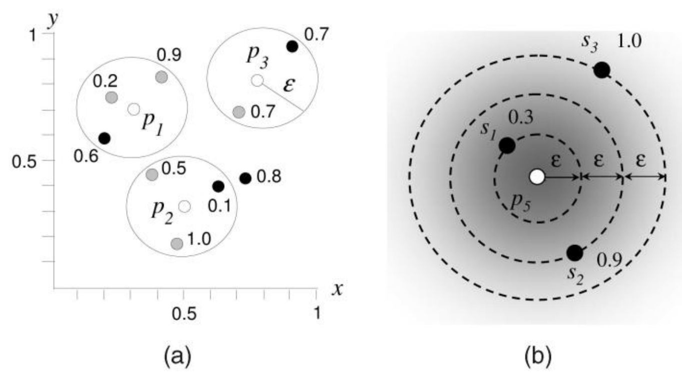


Fig. 1.Example of top-k spatial preference query a) Range Score b)Influence Score

The score T(p) of a hotel p is defined in terms of:

1) the maximum quality for each feature in the neighborhood region of p

2) the aggregation of those qualities.

The Range score, binds the neighborhood region to a circular region at p with radius ϵ (shown as a circle), and the aggregate function to SUM. For instance, the maximum quality of gray and black points within the circle of p1 are 0.9 and 0.6 respectively, so the score of p1 is T(p1)= 0.9+0.6=1.5. Similarly, we obtain T(p2)=1.0+0.1=1.1 and T(p3)=0.7+0.7=1.4. Hence,the hotel p1 is returned as the top result.

In fact, the semantics of the aggregate function is relevant to the user's query. The SUM function attempts to balance the overall qualities of all features. The neighborhood region in the above spatial preference query can also be defined by other score functions. A meaningful score function is the influence score (see Section 4).As opposed to the crisp radius ϵ constraint in the range score, the influence score smoothens the effect of ϵ and assigns higher weights to railway stations that are closer to the hotel. Fig. 1b shows a hotel p5 and three railway stations s_1, s_2, s_3 (with their quality values). The circles have their radii as multiples of ϵ .Now, the

ISSN: 2278-0181
Vol. 1 Issue 4, June - 2012

score of a railway station s_i is computed by multiplying its quality with the weight 2^{-j}, where j is the order of the smallest circle containing s_i. For example, the scores of s_1, s_2, and s_3 are $0.3*2^{-1}$=0.15, $0.9*2^{-2}$=0:225, and $1.0*2^{-3}$=0.125,respectively. The influence score of p5 is taken as the highest value (0.225).

Traditionally, there are two basic ways for ranking objects:

1) Spatial ranking, which orders the objects according to their distance from a reference point.
2) Non-spatial ranking, which orders the objects by an aggregate function on their non-. spatial values

Our top-k spatial preference query integrates these two types of ranking in an intuitive way. As indicated by our examples, this new query has a wide range of applications in service recommendation and decision support systems.

To our knowledge, there is no existing efficient solution for processing the top-k spatial reference query. A brute force approach (to be elaborated in Section 3.2) 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 data sets. In this paper, we propose alternative techniques that aim at minimizing the I/O accesses to the object and feature datasets, while being also computationally efficient. Specifically, we contribute the branch-and-bound (BB)algorithm for efficiently processing the top-k spatial preference query.

Furthermore, this paper studies one relevant extension that have not been investigated in our preliminary work [1].The extension (Section 3.4) is an optimized version of BB that exploits a more efficient technique for computing the scores of the objects. The second extension (Section 3.6) studies adaptations of the proposed algorithms for aggregate functions other than SUM, e.g., the functions MIN and MAX. The third extension (Section 4) develops solutions for the top-k spatial preference query based on the influence score. The rest of this paper is structured as follows: Section 2 provides background on basic and advanced queries on spatial databases, as well as top-k query evaluation in relational databases. Section 3 defines the top-k spatial preference query and presents our solutions. Section 4 studies the query extension for the influence score. In Section 5, our query algorithms are experimentally evaluated with real and synthetic data. Finally, Section 6 concludes the paper with future research directions.

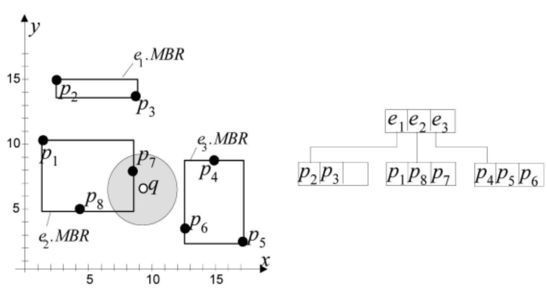
2. BACKGROUND AND RELATED WORK

Object ranking is a popular retrieval task in various applications. In relational databases, we rank tuples using an aggregate score function on their attribute values [2]. For example, a real estate agency maintains a database that contains information of flats available for sale. A potential customer wishes to view the top 10 flats with the largest sizes(area) and lowest prices. In this case, the score of each flat is expressed by the sum of two qualities: size and price, after normalization to the domain [0, 1] (e.g., 1 means the largest size and the lowest price and 0 means the smallest size and the highest price). In spatial databases, ranking is often associated to nearest neighbor (NN) retrieval. Given a query location, we are interested in retrieving the set of nearest objects to it that qualify a condition (e.g., restaurants).Assuming that the set of interesting objects is indexed by an R-tree [3], we can apply distance bounds and traverse the index in a branch-and-bound fashion to obtain the answer [4].

2.1 Spatial Query Evaluation on R-Trees

The most popular spatial access method is the R-tree [3], which indexes minimum bounding rectangles (MBRs) of objects. Fig. 2 shows a set D=(p1, . . . , p8) of spatial objects(e.g., points) and an R-tree that indexes them. R-trees can efficiently process main spatial query types, including spatial range queries, nearest neighbor queries, and spatial joins.

Given a spatial region W, a spatial range query retrieves from D the objects that intersect W. For instance, consider a range query that asks for all objects within the shaded area in Fig. 2. Starting from the root of the tree, the query is processed by recursively following entries, having MBRs that intersect the query region.



(a) (b)

Fig. 2. Spatial queries on R-trees. a) MBRs b)R-tree representation

For instance, e_1 does not intersect the query region, thus the subtree pointed by e_1 cannot contain any query result. In contrast, e_2 is followed by the algorithm and the points in the corresponding node are examined recursively to find the query result p_7.A nearest neighbor query takes as input a query object q and returns the closest object in D to q. For instance, the nearest neighbor of q in Fig. 2 is p_7. Its 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 of [4], provided that D is indexed by an R-tree. A min-heap H, which organizes R-tree entries based on the (minimum) distance of their MBRs to q is initialized with the root entries. In order to find the NN of q in Fig. 2, BF first inserts to H entries e_1,e_2, e_3, and their distances to q. Then, the nearest entry e_2 is retrieved from H and objects p1, p7, p8 are inserted to H. The next nearest entry in H is p7, which is the nearest neighbor of q. In terms of I/O, the BF algorithm is shown to be no worse than any NN algorithm on the same R-tree [4].

The aggregate R-tree (aR-tree) [10] is a variant of the Rtree, where each nonleaf entry augments an aggregate measure for some attribute value (measure) of all points in its subtree. As an example, the tree shown in Fig. 2 can be upgraded to a MAX aR-tree over the point set, if entries e1,e2,e3 contain the maximum measure values of sets (p2, p3), (p1,p8, p7), (p4, p5, p6), respectively. Assume that the measure values of p4,p5, p6 are 0.2,0.1, 0.4, respectively. In this case, the aggregate measure augmented in e3 would be max(0.2, 0.1,0.4) = 0.4. In this paper, we employ MAX aR-trees for indexing the feature data sets (e.g., restaurants),in order to accelerate the processing of top-k spatial preference queries.

Given a feature data set F and a multidimensional region R, the range top-k query selects the tuples (from F) within the region R and returns only those with the k highest qualities. Hong et al. [11] indexed the data set by a MAX aR-tree and developed an efficient tree traversal algorithm to answer the query. Instead of finding the best k qualities from F in a specified region, our (range score)query considers multiple spatial regions based on the points from the object data set D, and attempts to find out the best k regions (based on scores derived from multiple feature data sets Fc).

3 SPATIAL PREFERENCE QUERIES

Section 3.1 formally defines the top-k spatial preference query problem and describes the index structures for the data sets. Section 3.2 studies two baseline algorithms for processing the query. Section 3.3 presents an efficient branch-and-bound algorithm for the query, and its further optimization is proposed in Section 3.4. Section 3.5 develops a specialized spatial join algorithm for evaluating the query. Finally, Section 3.6 extends the above algorithms for answering top-k spatial preference queries involving other aggregate functions.

3.1 Definitions and Index Structures

Given an object data set D and m feature data sets F1,F2 . . . Fm, the top-k spatial preference query retrieves the k points in D with the highest score. Here, the overall score of an object point p ϵ D is defined as

$$T(p)=AGG\{T_c(p)|c \in [1,m]\} \qquad (1)$$

where AGG is an aggregate function (e.g: SUM,MIN,MAX etc)

$T_c(p)$ is the c^{th} component score of p with respect to the neighborhood condition and

m is the number of feature data sets.

The c^{th} component score of p i.e $T_c(p)$ can be computed as follows

$$T_c(p)=\max(\{w(s)|s \in F_c \wedge dist(p,s) \leq \epsilon\} \cup \{0\}). \qquad (2)$$

3.2 Algorithms

we develop various algorithms for processing top-k spatial preference queries. We first introduce a brute-force solution that computes the score of every point p ϵ D in order to obtain the query results. Then, we propose a group evaluation technique that computes the scores of multiple points concurrently.

3.2.1 Simple Probing Algorithm

For a point p ϵ D, where not all its component scores are known, its upper bound score $T_u(p)$ defined as

$$T_u(p)= \sum_{c=1}^{m} \begin{cases} T_c(p), & \text{if } T_c(p) \text{ is known} \\ 1, & \text{otherwise} \end{cases} \qquad (3)$$

It is guaranteed that the upper bound $T_u(p)$ is greater than or equals to the actual score T(p).

Algorithm 1 is a pseudocode of the simple probing (SP)algorithm, which retrieves the query results by computing the score of every object point. The algorithm uses two global variables: W_k is a min-heap for managing the top-k results and γ represents the top-k score so far (i.e., lowest score in W_k). Initially, the algorithm is invoked at the root node of the object tree (i.e., N =D.root). The procedure is recursively applied (at Line 4) on tree nodes until a leaf node is accessed. When a leaf node is reached, the component score $T_c(e)$ (at Line 8) is computed by executing a range search on the feature tree Fc for range score queries. Lines 6-8 describe an incremental computation technique, for reducing unnecessary component score computations. In particular, the point e is ignored as soon as its upper bound score $T_u(e)$ (see (3)) cannot be greater than the best-k score γ . The variables W_k and γ are updated when the actual score T(e) is greater than γ .

Algorithm 1. Simple Probing Algorithm
algorithm SP(Node N)
1: for each entry e $\in$ N do
2: If N is nonleaf then
3: read the child node N' pointed by e;
4: SP(N');
5: else
6: for c = 1 to m do
7: If $T_u(e)$ > γ then
//if upper bound is greater than γ
8: compute $T_c(p)$ using tree Fc; update $T_u(e)$;
9: If T(e) > γ then
10: update W_k and γ by e;

Drawbacks
1.it is very expensive because it comutes score for all objects.
2.No concurrency
3.it is not efficient method for larger input data sets.

3.2.2 Group Probing Algorithm

Due to separate score computations for different objects, SP is inefficient for large-object data sets. In view of this, we propose the group probing (GP) algorithm, a variant of SP,that reduces I/O cost by computing scores of objects in the same leaf node of the R-tree concurrently. 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 Fc tree.

We now introduce some distance notations for MBRs. Given a point p and an MBR e, the value mindist(p,e) [4] denotes the minimum possible distance between p and any point in e. Similarly, given two MBRs e_a and e_b, the value mindist(e_a, e_b) denotes the minimum possible distance between any point in ea and any point in eb.

Algorithm 2 shows the procedure for computing the c^{th} component score for a group of points. Consider a subset Vof D for which we want to compute their component score at feature tree Fc. Initially, the procedure is called with N being the root node of Fc. If e is a nonleaf entry and its mindist from some point p$\in$ V is within the range ϵ , then the procedure is applied recursively on the child node of e, since the subtree of Fc rooted at e may contribute to the component score of p. In case e is a leaf entry (i.e., a feature point), the scores of points in V are updated if they are within distance ϵ from e.

Algorithm 2. Group Probing Algorithm
algorithm GP(Node N, Set V , Value c,Value ϵ)
1: for each entry e $\in$ N do
2: If N is nonleaf then
3: If p $\in$ V , mindist(p,e) $\leq \epsilon$ then
4: read the child node N' pointed by e;
5: GP(N',V ,c, ϵ);
6: else
7: for each p $\in$ V such that dist(p,e) $\leq \epsilon$ do
8: $T_c(p)$=max{$T_c(p)$,w(e)};

Drawbacks
1.it is also expensive because it computes score for all objects but concurrently .

3.3 Branch-and-Bound Algorithm

GP is still expensive as it examines all objects in D and computes their component scores. We now propose an algorithm that can significantly reduce the number of objects to be examined. The key idea is to compute, for nonleaf entries e in the object tree D, an upper bound $T_u(p)$ of the score T(p) for any point p in the subtree of e. If $T_u(e) \leq \gamma$ then we need not access the subtree of e, thus we can save numerous score computations.

Algorithm 3 is a pseudocode of our BB algorithm, based on this idea. BB is called with N being the root node of D. If N is a nonleaf node, Lines 3-5 compute the scores T(e) for nonleaf entries e concurrently. Recall that $T_u(e)$ is an upperbound score for any point in the subtree of e.. If $T_u(e) \leq \gamma$,then the subtree of e cannot contain better results than those in Wk and it is removed from V . In order to obtain points with high scores early, we sort the entries in descending order of T(e) before invoking the above procedure recursively on the child nodes pointed by the entries in V .If N is a leaf node, we compute the scores for all points of N concurrently and then update the set Wk of the top-k results. Since both Wk and γ are global variables, their values are updated during recursive call of BB.

Algorithm 3. Branch-and-Bound Algorithm
W_k= new min-heap of size k (initially empty);
γ =0;
algorithm BB(Node N)
1: V ={e| e $\in$ N};
2: If N is nonleaf then

3: 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 $T_u(e) \leq \gamma$;
6: sort entries e $\in$ V in descending order of T(e);
7: for each entry e $\in$ V such that $T_u(e) > \gamma$ do
8: read the child node N' pointed by e;
9: BB(N');
10: else
11: for c = 1 to m do
12: compute $T_c(e)$ for all e $\in$ V concurrently;
13: remove entries e in V such that $T_u(e) \leq \gamma$;
14: update W_k and γ by entries in V ;

Advantages
1.it reduces number of objects to be examined.
2.it is efficient than SP and GP algorithms.

3.3.1 Upper Bound Score Computation

It remains to clarify how the (upper bound) scores Tc(p) of nonleaf entries (within the same node N) can be computed concurrently (at Line 4). Our goal is to compute these upperbound scores such that,

1).the bounds are computed with low I/O cost, and.

2).the bounds are reasonably tight, in order to facilitate effective pruning.

To achieve this, we utilize only level-1 entries (i.e., lowest level nonleaf entries) in Fc for deriving upper bound scores because:

1) there are much fewer level-1 entries than leaf entries (i.e., points)

2) high-level entries in Fc cannot provide tight bounds.

In our experimental study, we will also verify the effectiveness and the cost of using level-1entries for upper bound score computation. Algorithm 2 can be modified for the above upper bound computation task (where input V corresponds to a set of nonleaf entries), after changing Line 2 to check whether child nodes of N are above the leaf-level. The following example illustrates how upper bound range scores are derived. In Fig. 4a, v1 and v2 are nonleaf entries in the object tree D and the others are level-1 entries in the feature tree Fc. For the entry v1, we first define its Minkowski region [21] (i.e., gray region around v1), the area whose mindist from v1 is within ϵ . Observe that only entries e_i intersecting the Minkowski region of v_1 can contribute to the score of some point in v1. Thus, the upper bound score $T_c(p)$ is simply the maximum quality of entries e1,e5, e6, e7, i.e., 0.9. Similarly, $T_c(p)$ is computed as the maximum quality of entries e2, e3, e4, e8, i.e., 0.7. Assuming that v1 and v2 are entries in the same tree node of D, their upper bounds are computed concurrently to reduce I/O cost.

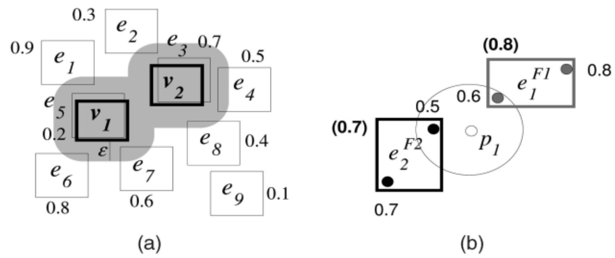


Fig. 4. Examples of deriving scores. (a) Upper bound scores. (b) Optimized computation.

3.4 Optimized Branch-and-Bound Algorithm

This section develops a more efficient score computation technique to reduce the cost of the BB algorithm.

3.4.1 Problem with BB Algorithm

Recall that Lines 11-13 of the BB algorithm are used to compute the scores of object points (i.e., leaf entries of the R-tree on D). A leaf entry e is pruned if its upper bound score $T_u(e)$ is not greater than the best score found so far γ . However, the upper bound score $T_u(e)$ (see (3)) is not tight because any unknown component score is replaced by 1.

Let us examine the computation of $T_u(p1)$ for the point p1 in Fig. 4b. The entry e_1^{F1} is a nonleaf entry from the feature tree F1. Its augmented quality value is $w(e_1^{F1})$=0.8. The entry points to a leaf node containing two feature points, whose qualities values are 0.6 and 0.8, respectively. Similarly, e_2^{F2} is a nonleaf entry from the tree F2 and it points to a leaf node of feature points.

Suppose that the best score found so far in BB is γ =1.4(not shown in the figure). We need to check whether the score of p1 can be higher than γ . For this, we compute the first component score T1(p1)= 0.6 by accessing the child node of e_1^{F1} . Now, we have the upper bound score of p1 as $T_u(p1)$ = 0.6 + 1.0= 1.6. Such a bound is above γ =1.4 so we need to compute the second component score T2(p1)= 0.5 by accessing the child node of e_2^{F2} . The exact score of p1 is T(p1)=0.6+0.5=1.1 the point p1 is then pruned because T(p1) $\leq \gamma$. In summary, two leaf nodes are accessed during the computation of T(p1) .

Our observation here is that the point p1 can be pruned earlier, without accessing the child node of e_2^{F2} . By taking the maximum quality of level-1 entries (from F2) that intersect the ϵ -range of p1, we derive: T2(p1) $\leq w(e_2^{F2})$=0.7.With the first component score T1(p1)=0.6, we infer that:T(p1)=

0.6 + 0.7= 1.3. Such a value is below γ so p1 can be pruned.

3.4.2 Optimized Computation of Scores

Based on our observation, we propose a tighter derivation for the upper bound score of p than the one shown in (3).Let p be an object point in D. Suppose that we have traversed some paths of the feature trees on F1,F2, . . . Fm.Let μ be an upper bound of the quality value for any unvisited entry (leaf or nonleaf) of the feature tree Fc. We then define the function $T_*(p)$ as

$$T_*(p)= \sum_{c=1}^{m} \max\left(\{w(s) \mid s \in F, dist(p,s) \leq \in, w(s) \geq \mu\} \cup \{\mu\}\right) \quad (4)$$

In the max function, the first set denotes the upper bound quality of any visited feature point within distance ϵ from p.According to (4), the value $T_*(p)$ is tight only when every c value is low. In order to achieve this, we access the feature trees in a round-robin fashion, and traverse the entries in each feature tree in descending order of quality values. Round-robin is a popular and effective strategy used for efficient merging of rankings [7], [9].

Algorithm 4 is the pseudocode for computing the scores of objects efficiently from the feature trees F1,F2,. . . ,Fm. The set V contains objects whose scores need to be computed. Here, ϵ refers to the distance threshold of the range score, and γ represents the best score found so far. Foreach feature tree Fc, we employ a max-heap Hc to traverse the entries of Fc in descending order of their quality values. The root of Fc is first inserted ino Hc. The variable μ maintains the upper bound quality of entries in the tree that will be visited. We then initialize each component score $T_c(p)$ of every object p ϵ V to 0.

Algorithm 4. Optimized Group Range Score Algorithm

algorithm Optimized_Group_Range(Trees F1;F2; . . . ;Fm, Set V , Value _, Value _)
1: for c := 1 to m do
2: Hc := new max-heap (with quality score as key);
3: insert Fc.root into Hc;
4: μ := 1;
5: for each entry p ϵ V do
6: $T_c(p)$:= 0;
7: α:= 1;
//ID of the current feature tree
8: while |V |> 0 and there exists a nonempty heap Hc do
9: deheap an entry e from H_α;
10: μ_α =w(e);
//update threshold
11: if ∀p ϵ V , mindist(p,e) > ϵ then
12: continue at Line 8;
13: for each p ϵ V do
// prune unqualified points
14: if ($\sum_{c=1}^{m} max\{\mu, Tc(p)\}$) ≤ γ then
15: remove p from V ;
16: read the child node CN pointed to by e;
17: for each entry e' of CN do
18: if CN is a nonleaf node then
19: if ∃p ϵ V , mindist(p,e') ≤ ϵ then
20: insert e' into H_α;
21: else
// update component scores
22: for each p ϵ V such that dist(p,e') ≤ ϵ do
23: T_α(p)=max{ T_α(p),w(e')};
24: α= next (round-robin) value where H_α is not empty;
25: for each entry p ϵ V do
26: T(p)=$\sum_{c=1}^{m} Tc(p)$;

At Line 7, the variable α keeps track of the ID of the current feature tree being processed. The loop at Line 8 is used to compute the scores for the points in the set V. We then deheap an entry e from the current heap H_α. The property of the max-heap guarantees that the quality value of any future entry deheaped from H_α is at most w(e). Thus, the bound μ is updated to w(e). At Lines 11-12, we prune the entry e if its distance from each object point p ϵ V is larger than ϵ . In case e is not pruned, we compute the tight upper bound score T(p) for each p ϵ V (by (4)); the object p is removed from V if T(p) ≤ γ (Lines 13-15).

Next, we access the child node pointed to by e, and examine each entry e' in the node (Lines 16-17). A nonleaf entry e' is inserted into the heap H_α if its minimum distance from some p ϵ V is within ϵ (Lines 18-20); whereas a leaf entry e' is used to update the component score $T_\alpha(p)$ for any p ϵ V within distance ϵ from e' (Lines 22-23). At Line 24, we apply the round-robin strategy to find the next α value such that the heap H_α is not empty. The loop at Line 8 repeats while V is not empty and there exists a nonempty heap Hc. At the end, the algorithm derives the exact scores for the remaining points of V .

3.4.3 The BB* Algorithm

Based on the above, we extend BB (Algorithm 3) to an optimized BB* algorithm as follows: First, Lines 11-13 of BB are replaced by a call to Algorithm 4, for computing the exact scores for object points in the set V . Second, Lines 3-5of BB are replaced by a call to a modified algorithm 4, for deriving the upper bound scores for nonleaf entries (in V).Such a modified Algorithm 4 is obtained after replacing Line 18 by checking whether the node CN is a nonleaf node above the level-1.

4. EXPERIMENTAL EVALUATION

In this section, we compare the efficiency of the proposed algorithms using real and synthetic data sets. Each data set is indexed by an aR-tree with 4 K bytes page size. We used an LRU memory buffer whose default size is set to 0.5 percent of the sum of tree sizes (for the object and feature trees used).Our algorithms were implemented in C++ and experiments were run on a Pentium D 2.8 GHz PC with 1 GB of RAM. In all experiments, we measure both the I/O cost (in number of page faults) and the total execution time (in seconds) of our algorithms.

5 RESULTS

In this section, we conduct experiments on real object and feature data sets in order to demonstrate the application of top-k spatial preference queries. We obtained three real spatial data sets from a travel portal,http://www.allstays.com/. Locations in these data sets correspond to (longitude and latitude)coordinates in US. We cleaned the data sets by discarding records without longitude and latitude. In summary, the relative performance between the algorithms in all experiments is consistent to the results on synthetic data.

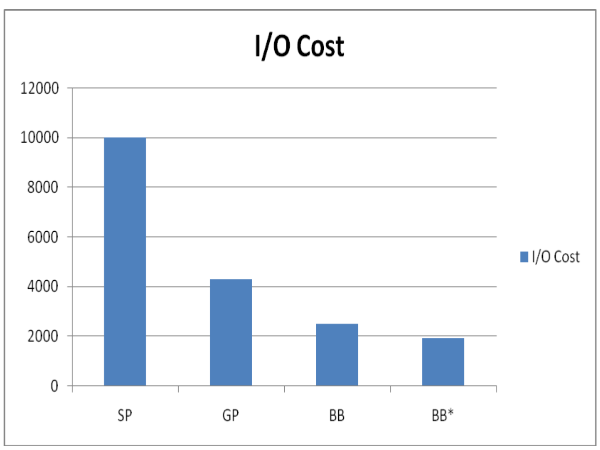


Fig. –Comparision of I/O cost for SP,GP,BB,BB*.

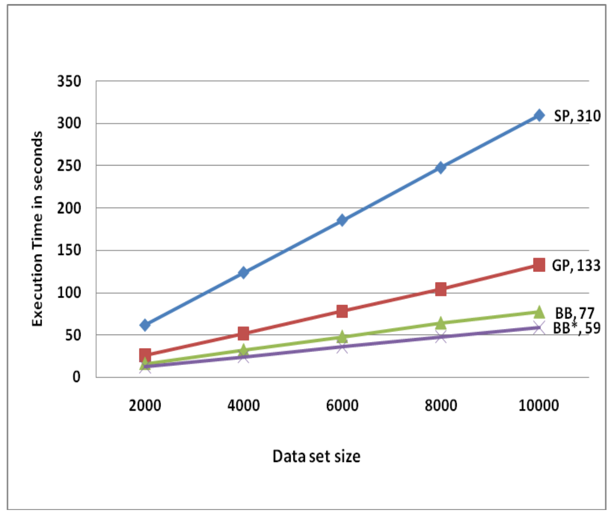


Fig.-Comparision of Execution Times for SP,GP,BB,BB*

6 CONCLUSION

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. The neighborhood of an object p 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 four algorithms for processing top-k spatial preference queries. The baseline algorithm SP computes the scores of every object by querying on feature data sets. The algorithm GP is a variant of SP that reduces I/O cost by computing scores of objects in the same leaf node concurrently. The algorithm BB derives upper bound scores for non leaf entries in the object tree, and prunes those that cannot lead to better results. The algorithm BB*is a variant of BB that utilizes an optimized method for computing the scores of objects (and upper bound scores of non leaf entries). Based on our experimental findings, BB* is scalable to large data sets and it is the most robust algorithm with respect to various parameters. In the future, we will study the top-k spatial preference query on a road network, in which the distance between two points is defined by their shortest path distance rather than their euclidean distance. The challenge is to develop alternative methods for computing the upper bound scores for a group of points on a road network.

.7.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 Queriesover Web-Accessible Databases," Proc. IEEE Int'l Conf. Data Eng.(ICDE), 2002.
[3] A. Guttman, "R-Trees: A Dynamic Index Structure for SpatialSearching," Proc. ACM SIGMOD, 1984.
[4] G.R. Hjaltason and H. Samet, "Distance Browsing in SpatialDatabases," ACM Trans. Database Systems, vol. 24, no. 2, pp. 265-318, 1999.
[5] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis andPerformance 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. DatabaseTheory (ICDT), 1999.
[7] R. Fagin, A. Lotem, and M. Naor, "Optimal AggregationAlgorithms for Middleware," Proc. Int'l Symp. Principles ofDatabase Systems (PODS), 2001.
[8] I.F. Ilyas, W.G. Aref, and A. Elmagarmid, "Supporting Top-k JoinQueries in Relational Databases," Proc. 29th Int'l Conf. Very LargeData Bases (VLDB), 2003.
[9] N. Mamoulis, M.L. Yiu, K.H. Cheng, and D.W. Cheung, "EfficientTop-k Aggregation of Ranked Inputs," ACM Trans. DatabaseSystems, vol. 32, no. 3, p. 19, 2007.
[10] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, "Efficient OLAPOperations in Spatial Data Warehouses," Proc. Int'l Symp. Spatialand Temporal Databases (SSTD), 2001.
[11] S. Hong, B. Moon, and S. Lee, "Efficient Execution of Range Top-kQueries in Aggregate R-Trees," IEICE Trans. Information andSystems, vol. 88-D, no. 11, pp. 2544-2554, 2005.
[12] T. Xia, D. Zhang, E. Kanoulas, and Y. Du, "On Computing Top-tMost Influential Spatial Sites," Proc. 31st Int'l Conf. Very Large DataBases (VLDB), 2005.
[13] Y. Du, D. Zhang, and T. Xia, "The Optimal-Location Query," Proc.Int'l Symp. Spatial and Temporal Databases (SSTD), 2005.
[14] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progessive Computation ofThe Min-Dist Optimal-Location Query," Proc. 32nd Int'l Conf. VeryLarge Data Bases (VLDB), 2006.
[15] Y. Chen and J.M. Patel, "Efficient Evaluation of All-Nearest-Neighbor Queries," Proc. IEEE Int'l Conf. Data Eng. (ICDE), 2007.
[16] P.G.Y. Kumar and R. Janardan, "Efficient Algorithms for ReverseProximity Query Problems," Proc. 16th ACM Int'l Conf. Advances inGeographic Information Systems (GIS), 2008.
[17] M.L. Yiu, P. Karras, and N. Mamoulis, "Ring-Constrained Join:Deriving Fair Middleman Locations from Pointsets via aGeometric Constraint," Proc. 11th Int'l Conf. Extending Database Technology (EDBT), 2008.
[18] M.L. Yiu, N. Mamoulis, and P. Karras, "Common Influence Join:A Natural Join Operation for Spatial Pointsets," Proc. IEEE Int'lConf. Data Eng. (ICDE), 2008.
[19] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient QueryProcessing in Geographic Web Search Engines," Proc. ACMSIGMOD, 2006.
[20] V.S. Sengar, T. Joshi, J. Joy, S. Prakash, and K. Toyama, "RobustLocation Search from Text Queries," Proc. 15th Ann. ACM Int'lSymp. Advances in Geographic Information Systems (GIS), 2007.
[21] S. Berchtold, C. Boehm, D. Keim, and H. Kriegel, "A Cost Modelfor Nearest Neighbor Search in High-Dimensional Data Space,"Proc. ACM Symp. Principles of Database Systems (PODS), 1997.
[22] E. Dellis, B. Seeger, and A. Vlachou, "Nearest Neighbor Search on Vertically Partitioned High-Dimensional Data," Proc. Seventh Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK), pp. 243-253, 2005.
[23] N. Mamoulis and D. Papadias, "Multiway Spatial Joins," ACMTrans. Database Systems, vol. 26, no. 4, pp. 424-475, 2001.
[24] A. Hinneburg and D.A. Keim, "An Efficient Approach to Clustering in Large Multimedia Databases with Noise," Proc.Fourth Int'l Conf. Knowledge Discovery and Data Mining (KDD), 1998.