Research Inventy: International Journal Of Engineering And Science
Vol.3, Issue 5 (July 2013), Pp 47-53
Issn(e): 2278-4721, Issn(p):2319-6483, Www.Researchinventy.Com

"Web Based Spatial Ranking System"

[1]Mr. Vijayakumar Neela, [2]Prof. Raafiya Gulmeher
(Dept. of Computer Science and Engineering, KBNCE, Gulbarga /VTU Belgaum, India)

***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 lease, 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, cafes, hospital, market, etc.) within their spatial neighborhood. Such a neighborhood concept can be specified by the user via different functions. It can be an explicit circular region within a given distance from the flat. Another intuitive definition is to assign higher weights to the features based on their proximity to the flat. In this paper, formally define spatial preference queries and propose appropriate indexing techniques and search algorithms for them. Extensive evaluation of this methods on both real and synthetic data reveals that an optimized branch-and-bound solution is efficient and robust with respect to different parameters.*

***KEYWORDS** -* Query processing, spatial databases

I. 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, they have presented an interesting type of preference queries, which select the best spatial location with respect to the quality of facilities in its spatial neighborhood. The 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.
Traditionally, there are two basic ways for ranking objects:

- Spatial ranking, which orders the objects according to their distance from a reference point.
- 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. As indicated by our examples, this new query has a wide range of applications in service recommendation and decision support systems. To gain knowledge yet, there is no existing efficient for processing the top-k spatial preference query. A bruteforce 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 data sets. In this paper, this propose alternative techniques that aim at minimizing the accesses to the object and feature data sets, while being also computationally efficient. This technique applies 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. Specifically, this contribute the branch-and-bound (BB) algorithm and the feature join (FJ) algorithm for efficiently processing the top-k spatial preference query. Furthermore, this paper studies three relevant extensions that have not been investigated in our preliminary work [1]. The first extension is an optimized version of BB that exploits a more efficient technique for computing the scores of the objects. The second extension studies adaptations of the proposed algorithms for aggregate functions other than SUM, e.g., the functions MIN and MAX. The third extension develops solutions for the top-k spatial preference query based on the influence score. For example, consider a real estate agency office that holds a database with available flats for lease. 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, cafes, hospital, market, etc.) in the spatial neighborhood of the flat (defined by a spatial range around it). Quality may be subjective and query-parametric. As another example, the user (e.g., a tourist) wishes to find a hotel p that is close to a high-quality restaurant and a high quality cafe.

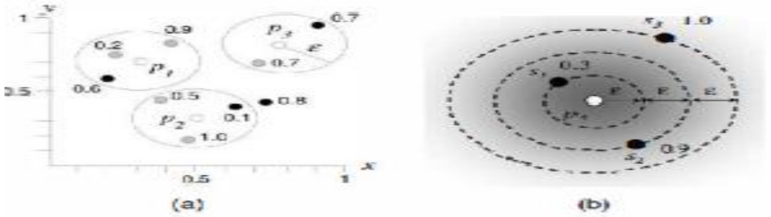


Fig. 1 Examples of top-k spatial preference queries.
(a)Range score (b) Influence score

Fig. la shows the locations of an object data set D (hotels) in white, and two feature data sets: the set J I (restaurants) in gray, and the set J 2 (cafes) in black. Feature points are labeled by quality values that can be obtained from rating providers (e.g., http://www.zagat.coml). For the ease of discussion, the qualities are normalized to values in [0,1]. The score T(P) of a hotel p is defined in terms of:
• The maximum quality for each feature in the neighborhood region of p.
• The aggregation of those qualities

A simple score instance, called the range score, binds the neighborhood region to a circular region at p with radius E (shown as a circle), and the aggregate function to SUM. For instance, the maximum quality of gray and black points within the circle of PI are 0.9 and 0.6, respectively, so the score PI is T(Pa =0.9 + 0.6 = l.5.Similarly,we obtain T(P2) =1.0 + 0.1 =1.1 and r(p3) = 0.7 + 0.7 = l A. Hence, the hotel p, is returned as the top result. 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. For the MIN function, the top result becomes P3, with the score T(P3) = min{ 0.7, 0.7} = 0.7. It ensures that the top result has reasonably high qualities in all features. For the MAX function, the top result is p2, with score of r(p2) = max{ 1.0, 0.1} = 1.0. It is used to optimize the quality in a particular feature, but not necessarily all of them. The neighborhood region in the above spatial preference query can also be defined by other score functions that is the influence score. As opposed to the crisp radius E constraint in the range score, the influence score smoothens the effect of E and assigns higher weights to cafes that are closer to the hotel.

Fig. 1 b shows a hotel P5 and three cafes s" S2, S3 (its quality with the weight 2·j, where j is the order of the smallest circle containing Sf. For example, the scores of s" S2 and S3 are0.3 12' = 0.15,0.9 1 22 = 0.225 and 1.0 / 2 3 = 0.125,respectively. The influence score of P5 is taken as the highest value (0.225). Furthermore, this paper studies three relevant extensions that have not been investigated in this preliminary work. The first extension is an optimized version of BB that exploits a more efficient technique for computing the scores of the objects.

The second extension studies adaptations of the proposed algorithms for aggregate functions other than SUM, e.g., the functions MIN and MAX. The third extension develops solutions for the top-k spatial preference query based on the influence score.

II. RELATED WORK

Object ranking is a popular retrieval task in various applications. In relational databases, it rank tuples using an aggregate score function on their attribute values. For example, a real estate agency maintains a database that contains information of flats available for rent. A potential customer wishes to view the top 10 flats with the largest sizes 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). In spatial databases, ranking is often associated to nearest neighbor (NN) retrieval. Given a query location, this is 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, this can apply distance bounds and traverse the index in a branch-and-bound fashion to obtain the answer.

Nevertheless, it is not always possible to use multidimensional indexes for top-k retrieval. First, such indexes break down in high-dimensional spaces. Second, top-k queries may involve an arbitrary set of user-specified attributes (e.g., size and price) from possible ones (e.g., size, price, distance to the beach, number of bedrooms, floor, etc.) 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. Solutions for top-k queries [2] focus on the efficient merging of object rankings that may arrive from different (distributed) sources. Their motivation is to minimize the number of accesses to the input rankings until the objects with the top k aggregate scores have been identified. To achieve this, upper and lower bounds for the objects seen so far are maintained while scanning the sorted lists. The first review of R-tree, which is the most popular spatial access method and the NN search algorithm of [4]. Then, it survey recent research of feature based spatial queries.

A. Special Query Evaluation on R-Tree

The most popular spatial access method is the R-tree, which indexes minimum bounding rectangles (MBRs) of objects can no longer be used to prune a combination based on distances among the entries in the combination. Any possible combination must be considered if its upper bound score is above the best score found.

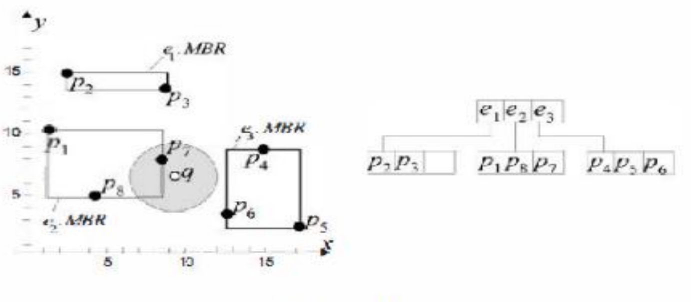


Fig. 2. Spatial queries on R-trees

Fig. 2 shows a set D = { Pl ... 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. For instance, el does not intersect the query region, thus the sub tree pointed by el cannot contain any query result. In contrast, e2 is followed by the algorithm and the points in the corresponding node are examined recursively to find the query result P7.

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 P7. 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 el, e2 , e3 and their distances to q. Then, the nearest entry is e2 retrieved from H and objects P1, P2, P3 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 (a R-tree) [6] 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 sub-tree. As an example, the tree shown in Fig. 2 can be upgraded to a MAX a R-tree over the point set, if entries el.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, Ps, P6 are {0.2, 0.l,0.4}, respectively. In this case, the aggregate measure augmented in e3 would be MAX{0.2, 0.l,0.4}=0.4. In this paper, we employ MAX a R-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 in [7]. It indexed the data set by a MAX a R-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).

B. Feature-Based Spatial Queries

It solved the problem of finding top-k sites based on their influence on feature points

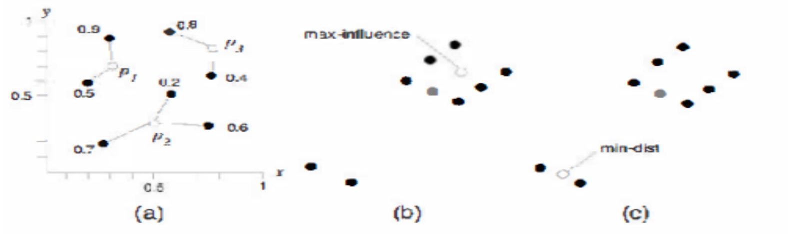


Fig. 3. Influential sites and optimal location queries. (a) Top-k influential. (b) Max-influence. (c) Min-distance.

.

As an example, Fig. 3a shows a set of sites (white points), a set of features (black points weights), such that each line links a feature point to its nearest site. The influence of a site PI is defined by the sum of weights of feature points having pi as their closest site. For instance, the score of PI is 0.9+0.5=1.4. Similarly, the scores of P2 and P3 are 1.5 and 1.2, respectively. Hence, p2 is returned as the top-l influential site.

Related to top-k influential sites query are the optimal location queries studied in [8] and [9]. The goal of the top k is to find the location in space (not chosen from a specific set of sites) that minimizes an objective function.

Fig. 3b and 3c shows, feature points and existing sites are shown as black and gray points, respectively. Assume that all feature points have the same quality. The maximum influence optimal location query [8] finds the location (to insert to the existing set of sites) with the maximum influence, whereas the minimum distance optimal location query [9] 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 in Fig. 3b and 3 c, respectively. The techniques proposed in [8] and [9] are specific to the particular query types described above and cannot be extended for our top-k spatial preference queries. Also, they deal with a single feature data set whereas our queries consider multiple feature data sets. Recently, novel spatial queries and joins [10] and [11], have been proposed for various spatial decision support problems. However, they do not utilize non-spatial qualities of facilities to define the score of a location. Finally, [12] and [13] studied the evaluation of textual location-based queries on spatial objects.

III. SPATIAL PREFERENCE QUERIES

A. Definitions and Index Structures

Let Fc be a feature data set, in which each feature object S ∈ Fc is associated with a quality w(s) and a spatial point. It assume that the domain of w(s) is the interval [0,1]. As an example, the quality w(s) of a restaurant s can be obtained from a ratings provider. It proceeds to elaborate the aggregate function and the component score function. Typical examples of the aggregate function AGG are: SUM, MIN, and MAX.

This first focus on the case where AGG is SUM. This will discuss the generic scenario where AGG is an arbitrary monotone aggregate function.

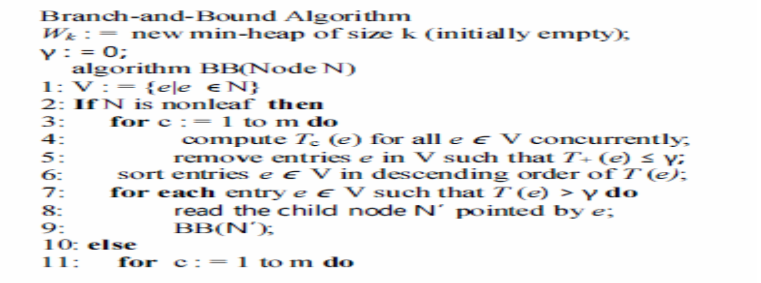
In this paper, It assume that the object data set . . D is indexed by an R-tree and each feature data set J e is indexed by an MAX a R-tree, where each non-leaf entry augments the maximum quality (of features) in its sub-tree. Nevertheless, this solution is directly applicable to data sets that are indexed by other hierarchical spatial indexes (e.g., point quad-trees).

The rationale of indexing different feature data sets by separate a R-trees is that:

- A user queries for only few features (e.g., restaurants and cafes) out of all possible features (e.g., restaurants, cafes, hospital, market, etc.).
- Different users may consider different subsets of features.

Branch-and-Bound algorithm 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) of the score T(e) for any point p in the subtree of e. If T(e) then we need not access the subtree of e, thus we can save numerous score computations.

This a pseudo code of BB algorithm, based on this idea. BB is called with N being the root node of D. If N is a non-leaf node, Lines 3 -5 compute the scores T(e) for non-leaf entries e concurrently. Recall that T(e) is an upper bound score for any point in the subtree of e. The techniques for computing T(e) will be discussed shortly with the component scores Te(e) known so far, it can derive T(e), an upper bound of T(e). If T(e) , 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 Y are global variables, their values are updated during recursive call of BB.



```
Branch-and-Bound Algorithm
$W_k$ : = new min-heap of size k (initially empty);
$\gamma$ : = 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 $T_+(e) \le \gamma$;
6:      sort entries $e \in$ V in descending order of $T(e)$;
7:     for each entry $e \in$ V such that $T(e) > \gamma$ 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 ∈ V concurrently;
13:         remove entries e in V such that $T_+(e) \leq \gamma$ ;
14:     update $W_k$ (and γ) by entries in V ;.
```

B.1 Upper Bound Score Computation

Upper Bound Score remains to clarify how the (upper bound) scores Tc(e) of non-leaf entries (within the same node N) can be computed concurrently . The goal is to compute these upper bound scores such that

The bounds are computed with low I/O cost

The bounds are reasonably tight, in order to facilitate effective pruning.

It utilizes only level-l entries (i.e., lowest level non leaf entries) in J e for deriving upper bound scores because:

1. There are much fewer level-l entries than leaf entries (i.e., points)
2. High-level entries in J e cannot provide tight bounds.

It also verifies the effectiveness and the cost of using level- 1 entries 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 non-leaf 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, v I and V2 are non-leaf entries in the object tree D and the others are level-l entries in the feature tree Fc' For the entry V1, we first define its Minkowski region (i.e., gray region around v I), the area whose mindist from V1 is within E. Observe that only entries e intersecting the Minkowski region of V1 can contribute to the scope of some point in V1. Thus, the upper bound score Te(v1) is simply the maximum quality of entries e1,.e8 ,e6, e7 i.e., 0.9. Similarly, Tc(v2) 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.

B. Optimized Branch-and-Bound Algorithm

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, E refers to the distance threshold of the range score, and Y represents the best score found s far. For each feature tree Fe, we employ a max-heap H c to traverse the entries of Fc in descending order of their quality values. The root of Fc is first inserted into H c. The variable μ_c maintains the upper bound quality of entries in the tree that will be visited. We then initialize each component score Tc(P) of every object p E V to O. The variable α keeps track of the ID of the current feature tree being processed. The loop is used to compute the scores for the points in the set.

C. Optimized Branch-and-Bound Algorithm

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, E refers to the distance threshold of the range score, and y represents the best score founds far. For each feature tree Fe, we employ a max-heap H c to traverse the entries of J e in descending order of their quality values. The root of Fc is first inserted into H c. The variable μ_c maintains the upper bound quality of entries in the tree that will be visited. We then initialize each component score Tc(P) of every object p E V to O. The variable α keeps track of the ID of the current feature tree being processed. The loop is used to compute the scores for the points in the set.

```
Optimized Group Range Score Algorithm
algorithm Optimized_Group_Range(Trees
$\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, Set V , Value ε, Value γ)
1: for  c := 1 to m do
2:      $H_c$ := new max-heap (with quality score as key);
3:      insert $\mathcal{F}_c$. root into $H_c$;
4:      $\mu_c$ := 1;
5:     for each entry $p \in V$ do
6:             $T_c(p)$ := 0;
7: α := 1;
8: while |V| > 0 and there exists a nonempty heap $H_c$ do
9:     deheap an entry e from $H_\alpha$ ;
10:    $\mu_\alpha$ := $w(e)$ ;
11:    if ∀ p∈V, mindist(p,e)>ε then
12:            continue at Line 8;
13:    for each $p \in V$ do
14:            if $\Sigma^m_{c=1}$ { $\mu_c$ , $T_c(p)$ } ≤ 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 ∈ V, mindist(p,e') ≤ ε then
20: insert e' into $H_\alpha$;
21: else
22:            for each $p \in V$ such that dist(p,e') ≤ ε do
23:                    $T_\alpha(p)$ := max{ $T_\alpha(p)$ , $w(e')$ };
24:            α := next (round-robin) value where $H_\alpha$ is not
               empty;
25: for each entry p 2 V do
26:            $T(p)$ := $\Sigma^m_{c=1}$ $T_c(p)$ ;
```

And then deheap an entry e from the current heap Hα . The property of the max-heap guarantees that the quality value of any future entry deheaped from Hα is at most w(e). Thus, the bound μc is updated to w(e). It prune the entry e if its distance from each object point pα V is larger than α. In case e is not pruned, it compute the tight upper bound score T*(p) for each pαV ; the object p is removed from V if T*(p) <= y.

Next, it access the child node pointed to bye, and examine each entry e ' in the node. A nonleaf entry e' is inserted into the heap Ha if its minimum distance from some p α V is within α whereas a leaf entry e' is used to update the component score Tα(P) for any p α V within distance E and it apply the round-robin strategy to find the next a value such that the heap Hα is not empty.

D. Influence Score

The influence score function that combines both the qualities and relative locations of feature points. And then it presents the adaptations of our solutions for the influence score function. Finally, it discuss how our solutions can be used for other types of influence score functions. The range score has a drawback that the parameter E is not easy to set. The range score has a drawback that the parameter α is not easy to set. Consider, for instance, the example of the range score T^{mg}( ). A score function such that it is not too sensitive to the range parameter α.The user usually prefer a high-quality restaurant rather than a large number of low-quality restaurants.

IV. EXPERIMENTS

The efficiency of the proposed algorithms using real and synthetic data sets is compared. Each data set is indexed by an R-tree with 4 K bytes page size. It 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). These algorithms were implemented in java and experiments were run on a Pentium D 2.8 GHz PC with 1 GB of RAM. In all experiments, it measure both the I/ O cost (in number of page faults) and the total execution time (in seconds) of this algorithms.

A. Experimental Settings

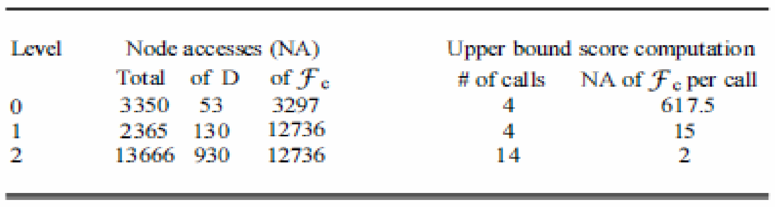
It used both real and synthetic data for the experiments. For each synthetic data set, the coordinates of points are random values uniformly and independently generated for different dimensions. By default, an object data set contains 200K points and a feature data set contains l00K points. The point coordinates of all data sets are normalized to the 2D space [0,10000f]^2 For a feature data set :Fc, we generated qualities for its points such that they simulate a real-world scenario: facilities close to (far from) a town centre often have high (low) quality. For this, a single anchor point s. is selected such that its neighborhood region contains high number of points. Let $dist_{min}$($dist_{max}$) be the minimum (maximum) distance of a point in :Fc from the anchor s.*.

B. Performance on Queries with Range Scores

It empirically justifies the choice of using level-l entries of feature trees Fc for the upper bound score computation routine in the BB algorithm. Table 1 shows the decomposition of node accesses over the tree D and the trees :Fc, and the statistics of upper bound score computation. Each accessed non-leaf node of D invokes a call of the upper bound score computation routine.

When level-0 entries of :Fc are used, each upper bound computation call incurs a high number (617.5) of node accesses (of :Fc). On the other hand, using level-2 entries for upper bound computation leads to very loose bounds, making it difficult to prune the leaf nodes of D. Observe that the total cost is minimized when level-l entries (of :Fc) are used. In that case, the node accesses per upper bound computation call is low (15), and yet the obtained bounds are tight enough for pruning most leaf nodes of D. The incremental computation technique derives a tight upper bound score (of each point) for the MIN function, a partially tight bound for SUM, and a loose bound for MAX . This explains the performance across different aggregate functions. However, the cost of the other methods is mainly influenced by the effectiveness of pruning. BB employs an effective technique to prune unqualified non-leaf entries in the object tree so it outperforms group probing.

TABLE 1
EFFECT OF THE LEVEL OF FC USED FOR UPPER BOUND SCORE COMPUTATION IN THE BB ALGORITHM



Level	Node accesses (NA)			Upper bound score computation	
	Total	of D	of $\mathcal{F}_c$	# of calls	NA of $\mathcal{F}_c$ per call
0	3350	53	3297	4	617.5
1	2365	130	12736	4	15
2	13666	930	12736	14	2

C Results on Real Data

This experiment on real object and feature data sets in order to demonstrate the application of top-k spatial preference queries. It obtained three real spatial data sets from a travel portal website, http://www.allstays.coml. Locations in these data sets correspond to (longitude and latitude) coordinates in US. This cleaned the data sets by discarding records without longitude and latitude. Each remaining location is normalized to a point in the 2D space $[0, l0,000]^2$ One data set is used as the object data set and the other two are used as feature data sets. The object data set D contains 11,399 camping locations. The feature data set :F1 ,contains 3 0,921 hotel records, each with a room price (quality) and a location. The feature data set: F2 has 3 ,848 records of Wal-Mart stores, each with a gasoline vailability (quality) and a location. The domain of each quality attribute (e.g., room price and gasoline availability) is normalized to the unit interval [0, 1]. Intuitively, a camping location is considered as good if it is close to a Wal-Mart store with high gasoline availability (i.e., convenient supply) and a hotel with high room price (which indirectly reflects the quality of nearby outdoor environment) this experiment, it can use the default parameter setting and study how the number of node accesses of BB is affected by the level of :Fc used.

V. CONCLUSIONS

The top-k spatial preference queries 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:
• The range score restricts the neighborhood to a crisp region centered at p,
• The influence score relaxes the neighborhood to the whole space and assigns higher weights to locations closer to p.

It presented above algorithms for processing top-k spatial preference queries. 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).
The algorithm performs a multiway join on feature trees to obtain qualified combinations of feature points and then search for their relevant objects in the object tree. Based on experimental findings, BB* is scalable to large data sets and it is the most robust algorithm with respect to various parameters. However, BB* is the best algorithm in cases where the number m of feature data sets is low and each feature data set is small.
Based on experimental findings, BB* is scalable to large data sets and it is the most robust algorithm with respect to various parameters. However, BB* is the best algorithm in cases where the number m of feature data sets is low and each feature data set is small.

REFERENCES

[1] M.L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis, "Top-k Spatia Preference Queries," Proc. IEEE Int"l Conf. Data Eng. (rCDE), 2007.
[2] N. Bruno, L. Gravano, and A. Marian, "Evaluating Top-k Queries over Web-Accessible Databases," Proc. IEEE Int"l Conf. Data Eng. (lCDE), 2002.
[3] Guttman, "R- Trees: A Dynamic Index Structure for Spatial Searching," Proc. ACM IGMOD, 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 HighDimensional Spaces," Proc. Int"l Conf. Very Large Data Bases (VLDB), 1998.