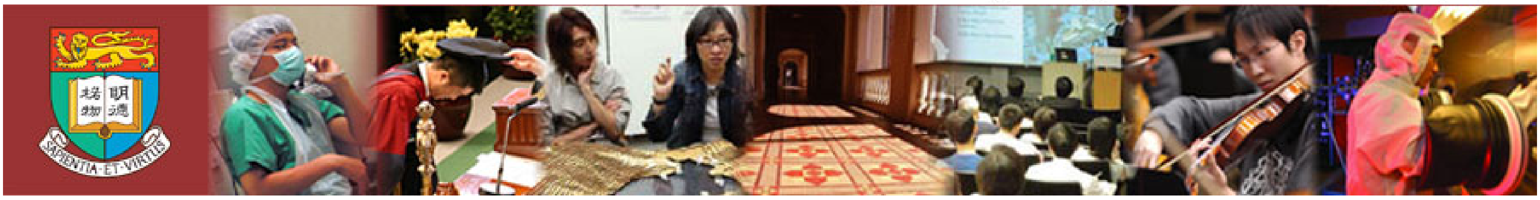


Ranking Spatial Data by Quality Preferences

Man Lung Yiu, Hua Lu, *Member*, *IEEE*, Nikos Mamoulis, and Michail Vaitis

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, 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 databases.

1 Introduction

Spatial database systems manage large collections of geographic entities, which apart from spatial attributes contain nonspatial 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 $\mathcal{D}$ of interesting *objects* (e.g., candidate locations), a top-k *spatial preference query* retrieves the k objects in $\mathcal{D}$ 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 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 nonspatial 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. For example, a user may define quality with respect to nonspatial attributes of restaurants around it (e.g., whether they serve seafood, price range, etc.).

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. 1a illustrates the locations of an object data set $\mathcal{D}$ (hotels) in white, and two feature data sets: the set $\mathcal{F}_1$ (restaurants) in gray, and the set $\mathcal{F}_2$ (cafes) in black. Feature points are labeled by quality values that can be obtained from rating providers (e.g., http://www.zagat.com/). For the ease of discussion, the qualities are normalized to values in $[0, 1]$. The score $\tau(p)$ of a hotel p is defined in terms of: 1) the maximum quality for each feature in the neighborhood region of p, and 2) 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 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 p_1 are 0.9 and 0.6, respectively, so the score of p_1 is $\tau(p_1) = 0.9 + 0.6 = 1.5$. Similarly, we obtain $\tau(p_2) = 1.0 + 0.1 = 1.1$ and $\tau(p_3) = 0.7 + 0.7 = 1.4$. Hence, the hotel p_1 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. For the MIN function, the top result becomes p_3, with the score $\tau(p_3) = \min\{0.7, 0.7\} = 0.7$. It ensures that the top result has reasonably high qualities in all features. For the MAX function, the top result is p_2, with $\tau(p_2) = \max\{1.0, 0.1\} = 1.0$. 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. 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 cafes that are closer to the hotel. Fig. 1b shows a hotel p_5 and three cafes s_1, s_2, s_3 (with their quality values). The circles have their radii as multiples of ϵ. Now, the score of a cafe 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 p_5 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, and 2) nonspatial ranking, which orders the objects by an

- *M. L. Yiu is with the Department of Computing, Hong Kong Polytechnic University, Hung Hom, Hong Kong. E-mail: csmlyiu@comp.polyu.edu.hk.*
- *H. Lu is with the Department of Computer Science, Aalborg University, DK-9220 Aalborg, Denmark. E-mail: luhua@cs.aau.dk.*
- *N. Mamoulis is with the Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong. E-mail: nikos@cs.hku.hk.*
- *M. Vaitis is with the Department of Geography, University of the Aegean, University Hill, GR-811 00 Mytilene, Greece. E-mail: vaitis@aegean.gr.*

Manuscript received 2 Feb. 2009; revised 9 Aug. 2009; accepted 21 Oct. 2009; published online 26 July 2010.
Recommended for acceptance by M. Ester.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-02-0046.
Digital Object Identifier no. 10.1109/TKDE.2010.119.

Published by the IEEE Computer Society

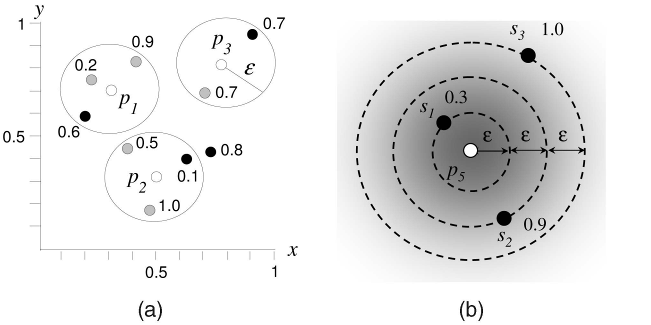


Fig. 1. Examples of top-k spatial preference queries. (a) Range score, $\epsilon = 0.2$ km. (b) Influence score, $\epsilon = 0.2$ km.

aggregate function on their nonspatial 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 preference query. A brute-force approach (to be elaborated in Section 3.2) for evaluating it is to compute the scores of all objects in $\mathcal{D}$ 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 data sets, while being also computationally efficient. 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. Specifically, we 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 (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 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, 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].

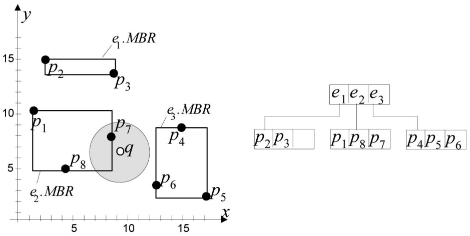


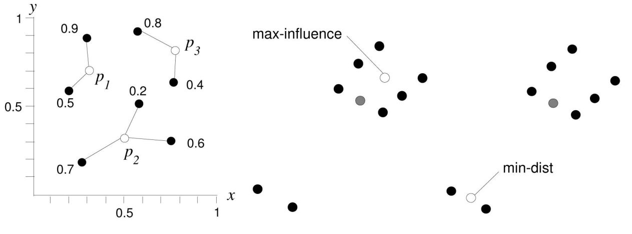
Fig. 2. Spatial queries on R-trees.

Nevertheless, it is not always possible to use multidimensional indexes for top-k retrieval. First, such indexes break down in high-dimensional spaces [5], [6]. 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 [7], [2], [8], [9] 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.

In the following sections, we first review the R-tree, which is the most popular spatial access method and the NN search algorithm of [4]. Then, we survey recent research of feature-based spatial queries.

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 $\mathcal{D} = \{p_1, \ldots, p_8\}$ 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 $\mathcal{D}$ 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, e_1 does not intersect the query region, thus the subtree pointed by e_1



(a) (b) (c)

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

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 $\mathcal{D}$ 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 $\mathcal{D}$ 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 p_1, p_7, p_8 are inserted to H. The next nearest entry in H is p_7, 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 R-tree, 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 e_1, e_2, e_3 contain the maximum measure values of sets $\{p_2, p_3\}, \{p_1, p_8, p_7\}, \{p_4, p_5, p_6\}$, respectively. Assume that the measure values of p_4, p_5, p_6 are $0.2, 0.1, 0.4$, respectively. In this case, the aggregate measure augmented in e_3 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 $\mathcal{F}$ and a multidimensional region R, the *range top-k query* selects the tuples (from $\mathcal{F}$) 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 $\mathcal{F}$ in a specified region, our (range score) query considers multiple spatial regions based on the points from the object data set $\mathcal{D}$, and attempts to find out the best k regions (based on scores derived from multiple feature data sets $\mathcal{F}_c$).

2.2 Feature-Based Spatial Queries

Xia et al. [12] solved the problem of finding top-k sites (e.g., restaurants) based on their *influence* on feature points (e.g., residential buildings). As an example, Fig. 3a shows a set of sites (white points), a set of features (black points with weights), such that each line links a feature point to its nearest site. The influence of a site p_i is defined by the sum of weights of feature points having p_i as their closest site. For instance, the score of p_1 is $0.9 + 0.5 = 1.4$. Similarly, the scores of p_2 and p_3 are 1.5 and 1.2, respectively. Hence, p_2 is returned as the top-1 influential site.

Related to top-k influential sites query are the optimal location queries studied in [13], [14]. The goal is to find the location in space (not chosen from a specific set of sites) that minimizes an objective function. In Figs. 3b and 3c, 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 [13] finds the location (to insert to the existing set of sites) with the maximum influence (as defined in [12]), whereas the minimum distance optimal location query [14] 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 Figs. 3b and 3c, respectively.

The techniques proposed in [12], [13], [14] 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 [15], [16], [17], [18] have been proposed for various spatial decision support problems. However, they do not utilize nonspatial qualities of facilities to define the score of a location. Finally, [19], [20] studied the evaluation of textual location-based queries on spatial objects.

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

Let $\mathcal{F}_c$ be a *feature* data set, in which each feature object $s \in \mathcal{F}_c$ is associated with a *quality* $\omega(s)$ and a spatial point. We assume that the domain of $\omega(s)$ is the interval $[0, 1]$. As an example, the quality $\omega(s)$ of a restaurant s can be obtained from a ratings provider.

Let $\mathcal{D}$ be an *object* data set, where each object $p \in \mathcal{D}$ is a spatial point. In other words, $\mathcal{D}$ is the set of interesting points (e.g., hotel locations) considered by the user.

Given an object data set $\mathcal{D}$ and m feature data sets $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, the *top-k spatial preference query* retrieves the k points in $\mathcal{D}$ with the highest score. Here, the score of an object point $p \in \mathcal{D}$ is defined as

$$\tau^\theta(p) = \mathtt{AGG}\left\{\tau_c^\theta(p) \mid c \in [1, m]\right\}, \quad (1)$$

TABLE 1
List of Notations

Notation	Meaning
e	an entry in an R-tree
$\mathcal{D}$	the object dataset
m	the number of feature datasets
$\mathcal{F}_c$	the c-th feature dataset
$\omega(s)$	the quality of an point s in $\mathcal{F}_c$
$\omega(e)$	augmented quality of an aR-tree entry e of $\mathcal{F}_c$
p	an object point of $\mathcal{D}$
$\tau_c^\theta(p)$	the c-the component score of p
$\tau^\theta(p)$	the overall score of p
$dist(p, s)$	Euclidean distance between two points p and s
$mindist(p, e)$	minimum distance between p and e
$maxdist(p, e)$	maximum distance between p and e
$\mathcal{T}(e)$	upper bound score of an R-tree entry e of $\mathcal{D}$

where AGG is an aggregate function and $\tau_c^\theta(p)$ is the (cth) component score of p with respect to the neighborhood condition θ and the (cth) feature data set $\mathcal{F}_c$.

We proceed to elaborate the aggregate function and the component score function. Typical examples of the aggregate function AGG are: SUM, MIN, and MAX. We first focus on the case where AGG is SUM. In Section 3.6, we will discuss the generic scenario where AGG is an arbitrary monotone aggregate function.

An intuitive choice for the component score function $\tau_c^\theta(p)$ is: the range score $\tau_c^{rng}(p)$, taken as the maximum *quality* $\omega(s)$ of points $s \in \mathcal{F}_c$ that are within a given parameter distance ϵ from p, or 0 if no such point exists.

$$\tau_c^{rng}(p) = \max(\{\omega(s) \mid s \in \mathcal{F}_c \wedge dist(p, s) \leq \epsilon\} \cup \{0\}). \quad (2)$$

In our problem setting, the user requires that an object $p \in \mathcal{D}$ must not be considered as a result if there exists some $\mathcal{F}_c$ such that the neighborhood region of p does not contain any feature point of $\mathcal{F}_c$.

There are other choices for the component score function $\tau_c^\theta(p)$. One example is the influence score function $\tau_c^{inf}(p)$ which will be considered in Section 4. Another example is the NN score $\tau_c^{nn}(p)$ that has been studied in our previous work [1], so it will not be examined again in this paper. The condition θ is dropped whenever the context is clear.

In this paper, we assume that the object data set $\mathcal{D}$ is indexed by an R-tree and each feature data set $\mathcal{F}_c$ is indexed by an MAX aR-tree, where each nonleaf entry augments the maximum quality (of features) in its subtree. Nevertheless, our solutions are 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 aR-trees is that: 1) a user queries for only few features (e.g., restaurants and cafes) out of all possible features (e.g., restaurants, cafes, hospital, market, etc.), and 2) different users may consider different subsets of features.

Based on the above indexing scheme, we develop various algorithms for processing top-k spatial preference queries. Table 1 lists the notations to be used throughout the paper.

3.2 Probing Algorithms

We first introduce a brute-force solution that computes the score of every point $p \in \mathcal{D}$ 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

According to Section 3.1, the quality $\omega(s)$ of any feature point s falls into the interval $[0, 1]$. Thus, for a point $p \in \mathcal{D}$, where not all its component scores are known, its upper bound score $\tau_+(p)$ is defined as

$$\tau_+(p) = \sum_{c=1}^{m} \begin{cases} \tau_c(p), & \text{if } \tau_c(p) \text{ is known,} \\ 1, & \text{otherwise.} \end{cases} \quad (3)$$

It is guaranteed that the bound $\tau_+(p)$ is greater than or equal to the actual score $\tau(p)$.

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., $N = \mathcal{D}.root$). 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 $\tau_c(e)$ (at Line 8) is computed by executing a range search on the feature tree $\mathcal{F}_c$ 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 $\tau_+(e)$ (see (3)) cannot be greater than the best-k score γ. The variables W_k and γ are updated when the actual score $\tau(e)$ 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 $\tau_+(e) > \gamma$ then          ▷ upper bound score
8:                  compute $\tau_c(e)$ using tree $\mathcal{F}_c$; update $\tau_+(e)$;
9:          If $\tau(e) > \gamma$ then
10:             update $W_k$ (and $\gamma$) by $e$;
```

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 $\mathcal{F}_c$ tree.

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

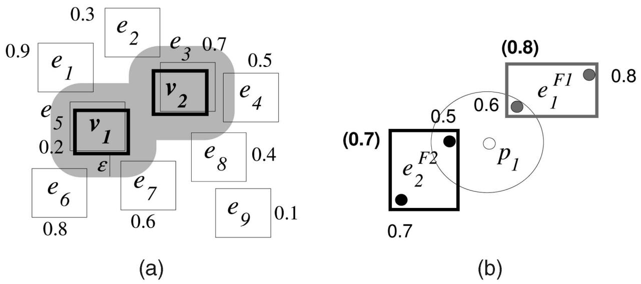


Fig. 4. Examples of deriving scores. (a) Upper bound scores. (b) Optimized computation.

($maxdist(e_a, e_b)$) denotes the minimum (maximum) possible distance between any point in e_a and any point in e_b.

Algorithm 2 shows the procedure for computing the cth component score for a group of points. Consider a subset V of $\mathcal{D}$ for which we want to compute their $\tau_c^{rng}(p)$ score at feature tree $\mathcal{F}_c$. Initially, the procedure is called with N being the root node of $\mathcal{F}_c$. 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 $\mathcal{F}_c$ 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 Range Score Algorithm

```
    algorithm Group_Range(Node $N$, Set $V$, Value $c$,
    Value $\epsilon$)
1:  for each entry $e \in N$ do
2:      If $N$ is nonleaf then
3:          If $\exists p \in V,\ mindist(p,e) \leq \epsilon$ then
4:              read the child node $N'$ pointed by $e$;
5:              Group_Range($N'$,$V$,$c$, $\epsilon$);
6:      else
7:          for each $p \in V$ such that $dist(p,e) \leq \epsilon$ do
8:              $\tau_c(p) := \max\{\tau_c(p), \omega(e)\}$;
```

3.3 Branch-and-Bound Algorithm

GP is still expensive as it examines all objects in $\mathcal{D}$ 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 $\mathcal{D}$, an *upper bound* $\mathcal{T}(e)$ of the score $\tau(p)$ for any point p in the subtree of e. If $\mathcal{T}(e) \leq \gamma$, 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 $\mathcal{D}$. If N is a nonleaf node, Lines 3-5 compute the scores $\mathcal{T}(e)$ for nonleaf entries e concurrently. Recall that $\mathcal{T}(e)$ is an upper bound score for any point in the subtree of e. The techniques for computing $\mathcal{T}(e)$ will be discussed shortly. Like (3), with the component scores $\mathcal{T}_c(e)$ known so far, we can derive $\mathcal{T}_+(e)$, an upper bound of $\mathcal{T}(e)$. If $\mathcal{T}_+(e) \leq \gamma$, then the subtree of e cannot contain better results than those in W_k and it is removed from V. In order to obtain points with high scores early, we sort the entries in descending order of $\mathcal{T}(e)$ 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 W_k of the top-k results. Since both W_k 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);
    $\gamma := 0$;                                  ▷ $k$th score in $W_k$

    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 $\mathcal{T}_c(e)$ for all $e \in V$ concurrently;
5:          remove entries $e$ in $V$ such that $\mathcal{T}_+(e) \leq \gamma$;
6:      sort entries $e \in V$ in descending order of $\mathcal{T}(e)$;
7:      for each entry $e \in V$ such that $\mathcal{T}(e) > \gamma$ do
8:          read the child node $N'$ pointed by $e$;
9:          BB($N'$);
10: else
11:     for $c := 1$ to $m$ do
12:         compute $\tau_c(e)$ for all $e \in V$ concurrently;
13:         remove entries $e$ in $V$ such that $\tau_+(e) \leq \gamma$;
14:     update $W_k$ (and $\gamma$) by entries in $V$;
```

3.3.1 Upper Bound Score Computation

It remains to clarify how the (upper bound) scores $\mathcal{T}_c(e)$ of nonleaf entries (within the same node N) can be computed concurrently (at Line 4). Our goal is to compute these upper bound scores such that

- the bounds are computed with low I/O cost, and
- 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 $\mathcal{F}_c$ for deriving upper bound scores because: 1) there are much fewer level-1 entries than leaf entries (i.e., points), and 2) high-level entries in $\mathcal{F}_c$ cannot provide tight bounds. In our experimental study, we will also verify 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 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, v_1 and v_2 are nonleaf entries in the object tree $\mathcal{D}$ and the others are level-1 entries in the feature tree $\mathcal{F}_c$. For the entry v_1, we first define its Minkowski region [21] (i.e., gray region around v_1), the area whose $mindist$ from v_1 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 v_1. Thus, the upper bound score $\mathcal{T}_c(v_1)$ is simply the maximum quality of entries e_1, e_5, e_6, e_7, i.e., 0.9. Similarly, $\mathcal{T}_c(v_2)$ is computed as the maximum quality of entries e_2, e_3, e_4, e_8, i.e., 0.7. Assuming that v_1 and v_2 are entries in the same tree node of $\mathcal{D}$, their upper bounds are computed concurrently to reduce I/O cost.

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 Motivation

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 $\mathcal{D}$). A leaf entry e is pruned if its upper bound score $\tau_+(e)$ is not greater than the best score found so far γ. However, the upper bound score $\tau_+(e)$ (see (3)) is not tight because any unknown component score is replaced by a loose bound (i.e., the value 1).

Let us examine the computation of $\tau_+(p_1)$ for the point p_1 in Fig. 4b. The entry e_1^{F1} is a nonleaf entry from the feature tree $\mathcal{F}_1$. Its augmented quality value is $\omega(e_1^{F1}) = 0.8$. 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 $\mathcal{F}_2$ and it points to a leaf node of feature points.

Suppose that the best score found so far in BB is $\gamma = 1.4$ (not shown in the figure). We need to check whether the score of p_1 can be higher than γ. For this, we compute the first component score $\tau_1(p_1) = 0.6$ by accessing the child node of e_1^{F1}. Now, we have the upper bound score of p_1 as $\tau_+(p) = 0.6 + 1.0 = 1.6$. Such a bound is above $\gamma = 1.4$ so we need to compute the second component score $\tau_2(p_1) = 0.5$ by accessing the child node of e_2^{F2}. The exact score of p_1 is $\tau(p_1) = 0.6 + 0.5 = 1.1$; the point p_1 is then pruned because $\tau(p_1) \leq \gamma$. In summary, two leaf nodes are accessed during the computation of $\tau(p_1)$.

Our observation here is that the point p_1 can be pruned earlier, without accessing the child node of e_2^{F2}. By taking the maximum quality of level-1 entries (from $\mathcal{F}_2$) that intersect the ϵ-range of p_1, we derive: $\tau_2(p_1) \leq \omega(e_2^{F2}) = 0.7$. With the first component score $\tau_1(p_1) = 0.6$, we infer that: $\tau(p_1) \leq 0.6 + 0.7 = 1.3$. Such a value is below γ so p_1 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 $\mathcal{D}$. Suppose that we have traversed some paths of the feature trees on $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$. Let μ_c be an upper bound of the quality value for any unvisited entry (leaf or nonleaf) of the feature tree $\mathcal{F}_c$. We then define the function $\tau_\star(p)$ as

$$\tau_\star(p) = \sum_{c=1}^{m} \max(\{\omega(s) \mid s \in \mathcal{F}_c, dist(p,s) \leq \epsilon, \omega(s) \geq \mu_c\} \cup \{\mu_c\}). \tag{4}$$

In the max function, the first set denotes the upper bound quality of any visited feature point within distance ϵ from p. The following lemma shows that the value $\tau_\star(p)$ is always greater than or equal to the actual score $\tau(p)$.

Lemma 1. *It holds that $\tau_\star(p) \geq \tau(p)$, for any $p \in \mathcal{D}$.*

Proof. If the actual component score $\tau_c(p)$ is above μ_c, then, $\tau_c(p) = \max\{\omega(s) \mid s \in \mathcal{F}_c, dist(p,s) \leq \epsilon, \omega(s) \geq \mu_c\}$. Otherwise, we derive $\tau_c(p) \leq \mu_c$. In both cases, we have

$$\tau_c(p) \leq \max(\{\omega(s) \mid s \in \mathcal{F}_c, dist(p,s) \leq \epsilon, \omega(s) \geq \mu_c\} \cup \{\mu_c\}).$$

Therefore, we have $\tau_\star(p) \geq \tau(p)$. □

According to (4), the value $\tau_\star(p)$ is tight only when every μ_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]. Alternative strategies include the selectivity-based strategy and the fractal-dimension strategy [22]. These strategies are designed specifically for coping with high-dimensional data, however, in our problem setting, they have insignificant performance gain over round-robin.

Algorithm 4 is the pseudocode for computing the scores of objects efficiently from the feature trees $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$. 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. For each feature tree $\mathcal{F}_c$, we employ a max-heap H_c to traverse the entries of $\mathcal{F}_c$ in descending order of their quality values. The root of $\mathcal{F}_c$ 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 $\tau_c(p)$ of every object $p \in V$ to 0.

Algorithm 4. Optimized Group Range Score Algorithm

```
    algorithm Optimized_Group_Range(Trees
    $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, Set $V$, Value $\epsilon$, Value $\gamma$)
 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:         $\tau_c(p) := 0$;
 7: $\alpha := 1$;                    ▷ ID of the current feature tree
 8: while $|V| > 0$ and there exists a nonempty heap $H_c$ do
 9:     deheap an entry $e$ from $H_\alpha$;
10:     $\mu_\alpha := \omega(e)$;                      ▷update threshold
11:     if $\forall p \in V, mindist(p,e) > \epsilon$ then
12:         continue at Line 8;
13:     for each $p \in V$ do          ▷ prune unqualified points
14:         if $(\sum_{c=1}^{m} \max\{\mu_c, \tau_c(p)\}) \leq \gamma$ 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 $\exists p \in V,\ mindist(p,e') \leq \epsilon$ then
20:                 insert $e'$ into $H_\alpha$;
21:         else                      ▷ update component scores
22:             for each $p \in V$ such that $dist(p,e') \leq \epsilon$ do
23:                 $\tau_\alpha(p) := \max\{\tau_\alpha(p), \omega(e')\}$;
24:     $\alpha$ := next (round-robin) value where $H_\alpha$ is not
        empty;
25: for each entry $p \in V$ do
26:     $\tau(p) := \sum_{c=1}^{m} \tau_c(p)$;
```

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 $\omega(e)$. Thus, the bound μ_c is updated to $\omega(e)$. At Lines 11-12, we prune the entry e if its distance from each object point $p \in V$ is larger than ϵ. In case e is not pruned, we compute the tight upper bound score $\tau_\star(p)$ for each $p \in V$ (by (4)); the object p is removed from V if $\tau_\star(p) \leq \gamma$ (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 $p \in V$ is within ϵ (Lines 18-20); whereas a leaf entry e' is used to update the component score $\tau_\alpha(p)$ for any $p \in V$ 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 H_c. 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 3-5 of 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.

3.5 Feature Join Algorithm

An alternative method for evaluating a top-k spatial preference query is to perform a multiway spatial join [23] on the feature trees $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$ to obtain combinations of feature points which can be in the neighborhood of some object from $\mathcal{D}$. Spatial regions which correspond to combinations of high scores are then examined, in order to find data objects in $\mathcal{D}$ having the corresponding feature combination in their neighborhood. In this section, we first introduce the concept of a combination, then discuss the conditions for a combination to be pruned, and finally elaborate the algorithm used to progressively identify the combinations that correspond to query results.

Tuple $\langle f_1, f_2, \ldots, f_m \rangle$ is a *combination* if, for any $c \in [1, m]$, f_c is an entry (either leaf or nonleaf) in the feature tree $\mathcal{F}_c$. The score of the combination is defined by

$$\tau(\langle f_1, f_2, \ldots, f_m \rangle) = \sum_{c=1}^{m} \omega(f_c). \tag{5}$$

For a nonleaf entry f_c, $\omega(f_c)$ is the MAX of all feature qualities in its subtree (stored with f_c, since $\mathcal{F}_c$ is an aR-tree). A combination disqualifies the query if

$$\exists\, (i \neq j \wedge i, j \in [1, m]),\ mindist(f_i, f_j) > 2\epsilon. \tag{6}$$

When such a condition holds, it is impossible to have a point in $\mathcal{D}$ whose $mindist$ from f_i and f_j are both within ϵ, respectively. The above validity check acts as a multiway join condition that significantly reduces the number of combinations to be examined.

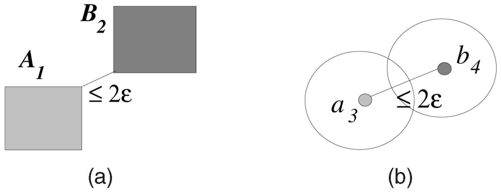


Fig. 5. Qualified combinations for the join. (a) Nonleaf combination. (b) Leaf combination.

Figs. 5a and 5b illustrate the condition for a nonleaf combination $\langle A_1, B_2 \rangle$ and a leaf combination $\langle a_3, b_4 \rangle$, respectively, to be a candidate combination for the query.

Algorithm 5 is a pseudocode of our feature join algorithm. It employs a max-heap H for managing combinations of feature entries in descending order of their combination scores. The score of a combination $\langle f_1, f_2, \ldots, f_m \rangle$ as defined in (5) is an upper bound of the scores of all combinations $\langle s_1, s_2, \ldots, s_m \rangle$ of feature points, such that s_c is located in the subtree of f_c for each $c \in [1, m]$. Initially, the combination with the root pointers of all feature trees is enheaped. We progressively deheap the combination with the largest score. If all its entries point to leaf nodes, then we load these nodes $L_1, \ldots, L_m$ and call Find_Result to traverse the object R-tree $\mathcal{D}$ and find potential results. Find_Result is a variant of the BB algorithm, with the following difference: $L_1, \ldots, L_m$ are viewed as m tiny feature trees (each with one node) and accesses to them incur no extra I/O cost.

Algorithm 5. Feature Join Algorithm

```
    $W_k$ := new min-heap of size $k$ (initially empty);
    $\gamma := 0$;                                  ▷$k$th score in $W_k$

    algorithm FJ(Tree $\mathcal{D}$,Trees $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$)
 1: $H$ := new max-heap (combination score as the key);
 2: insert $\langle \mathcal{F}_1.root, \mathcal{F}_2.root, \ldots, \mathcal{F}_m.root \rangle$ into $H$;
 3: while $H$ is not empty do
 4:     deheap $\langle f_1, f_2, \ldots, f_m \rangle$ from $H$;
 5:     if $\forall\, c \in [1, m]$, $f_c$ points to a leaf node
 6:         for $c := 1$ to $m$ do
 7:             read the child node $L_c$ pointed by $f_c$;
 8:         Find_Result($\mathcal{D}.root$, $L_1, \ldots, L_m$);
 9:     else
10:         $f_c$ := highest level entry among $f_1, f_2, \ldots, f_m$;
11:         read the child node $N_c$ pointed by $f_c$;
12:         for each entry $e_c \in N_c$ do
13:             insert $\langle f_1, f_2, \ldots, e_c, \ldots, f_m \rangle$ into $H$ if its score is
    greater than $\gamma$ and it qualifies the query;

    algorithm Find_Result(Node $N$, Nodes $L_1, \ldots, L_m$)
 1: for each entry $e \in N$ do
 2:     if $N$ is nonleaf then
 3:         compute $\mathcal{T}(e)$ by entries in $L_1, \ldots, L_m$;
 4:         if $\mathcal{T}(e) > \gamma$ then
 5:             read the child node $N'$ pointed by $e$;
 6:             Find_Result($N'$, $L_1, \ldots, L_m$);
 7:     else
 8:         compute $\tau(e)$ by entries in $L_1, \ldots, L_m$;
 9:         update $W_k$ (and $\gamma$) by $e$ (when necessary);
```

In case not all entries of the deheaped combination point to leaf nodes (Line 9 of FJ), we select the one at the highest level, access its child node N_c and then form new combinations with the entries in N_c. A new combination is inserted into H for further processing if its score is higher than γ and it qualifies the query. The loop (at Line 3) continues until H becomes empty.

3.6 Extension to Monotonic Aggregate Functions

We now extend our proposed solutions for processing the top-k spatial preference query defined by any monotonic aggregate function AGG. Examples of AGG include (but not limited to) the MIN and MAX functions.

3.6.1 Adaptation of Incremental Computation

Recall that the incremental computation technique is applied by algorithms SP, GP, and BB, for reducing I/O cost. Specifically, even if some component score $\tau_c(p)$ of a point p has not been computed yet, the upper bound score $\tau_+(p)$ of p can be derived by (3). Whenever $\tau_+(p)$ drops below the best score found so far γ, the point p can be discarded immediately without needing to compute the unknown component scores of p.

In fact, the algorithms SP, GP, and BB are directly applicable to any monotonic aggregate function AGG because (3) can be generalized for AGG. Now, the upper bound score $\tau_+(p)$ of p is defined as

$$\tau_+(p) = \mathtt{AGG}_{\,c=1}^{\,m} \begin{cases} \tau_c(p), & \text{if } \tau_c(p) \text{ is known,} \\ 1, & \text{otherwise.} \end{cases} \tag{7}$$

Due to the monotonicity property of AGG, the bound $\tau_+(p)$ is guaranteed to be greater than or equal to the actual score $\tau(p)$.

3.6.2 Adaptation of Upper Bound Computation

The BB* and FJ algorithms compute the upper bound score of a nonleaf entry of the object tree $\mathcal{D}$ or a combination of entries from feature trees, by summing its upper bound component scores. Both BB* and FJ are applicable to any monotonic aggregate function AGG, with only the slight modifications discussed below. For BB*, we replace the summation operator by AGG, in (4), and at Lines 14 and 26 of Algorithm 4. For FJ, we replace the summation by AGG, in (5).

4 Influence Score

This section first studies the *influence score* function that combines both the qualities and relative locations of feature points. It then presents the adaptations of our solutions in Section 3 for the influence score function. Finally, we discuss how our solutions can be used for other types of influence score functions.

4.1 Score Definition

The range score has a drawback that the parameter ϵ is not easy to set. Consider, for instance, the example of the range score $\tau^{rng}()$ in Fig. 6a, where the white points are object points in $\mathcal{D}$, the gray points and black points are feature points in the feature sets $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. If ϵ is set to 0.2 (shown by circles), then the object p_2 has the score $\tau^{rng}(p_2) = 0.9 + 0.1 = 1.0$ and it cannot be the best object (as $\tau^{rng}(p_1) = 1.2$). This happens because a high-quality black feature is barely outside the ϵ-range of p_2. Had ϵ been slightly larger, that black feature (with weight 0.6) would contribute to the score of p_2, making it the best object.

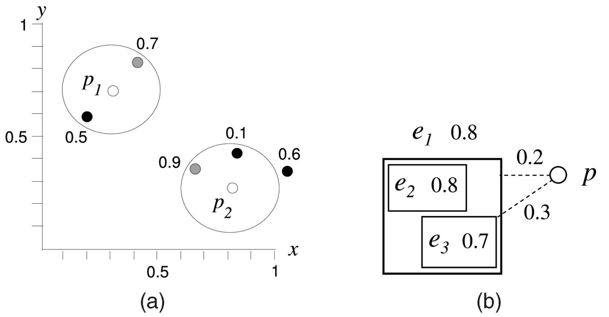


Fig. 6. Example of influence score ($\epsilon = 0.2$). (a) Exact score. (b) Upper bound property.

In the field of statistics, the Gaussian density function [24] has been used to estimate the density in the space, from a set $\mathcal{F}$ of points. The density at location p is estimated as: $\mathcal{G}(p) = \sum_{f \in \mathcal{F}} \exp(-\frac{dist^2(p,f)}{2\sigma^2})$, where σ is a parameter. Its advantage is that the value $\mathcal{G}(p)$ is not sensitive to a slight change in σ. $\mathcal{G}(p)$ is mainly contributed by the points (of $\mathcal{F}$) close to p and weakly affected by the points far away.

Inspired by the above function, we devise a score function such that it is not too sensitive to the range parameter ϵ. In addition, the users in our application usually prefer a high-quality restaurant (i.e., a feature point) rather than a large number of low-quality restaurants. Therefore, we use the maximum operator rather than the summation in $\mathcal{G}(p)$. Specifically, we define the *influence score* of an object point p with respect to the feature set $\mathcal{F}_c$ as

$$\tau_c^{inf}(p) = \max\{\, \omega(s) \cdot 2^{-\frac{dist(p,s)}{\epsilon}} \mid s \in \mathcal{F}_c \,\}, \tag{8}$$

where $\omega(s)$ is the quality of s, ϵ is a user-specified range, and $dist(p,s)$ is the distance between p and s.

The overall score $\tau^{inf}(p)$ of p is then defined as

$$\tau^{inf}(p) = \mathtt{AGG}\ \{\ \tau_c^{inf}(p) \mid c \in [1,m]\ \}, \tag{9}$$

where AGG is a monotone aggregate operator and m is the number of feature data sets. Again, we focus on the case where AGG is the SUM function.

Let us compute the influence score $\tau^{inf}()$ for the points in Fig. 6a, assuming $\epsilon = 0.2$. From Fig. 6a, we obtain

$$\begin{aligned} \tau^{inf}(p_1) &= \max\left\{0.7 \cdot 2^{-\frac{0.18}{0.20}}, 0.9 \cdot 2^{-\frac{0.50}{0.20}}\right\} \\ &\quad + \max\left\{0.5 \cdot 2^{-\frac{0.18}{0.20}}, 0.1 \cdot 2^{-\frac{0.60}{0.20}}, 0.6 \cdot 2^{-\frac{0.80}{0.20}}\right\} \\ &= 0.643 \text{ and} \\ \tau^{inf}(p_2) &= \max\left\{0.9 \cdot 2^{-\frac{0.18}{0.20}}, 0.7 \cdot 2^{-\frac{0.65}{0.20}}\right\} \\ &\quad + \max\left\{0.1 \cdot 2^{-\frac{0.19}{0.20}}, 0.6 \cdot 2^{-\frac{0.22}{0.20}}, 0.5 \cdot 2^{-\frac{0.70}{0.20}}\right\} = 0.762. \end{aligned}$$

The top-1 point is p_2, implying that the influence score can capture feature points outside the range $\epsilon = 0.2$. In fact, the influence score function possesses two nice properties. First, a feature point s that is barely outside the range ϵ (from the object point p) still has potential to contribute to the score, provided that its quality $\omega(s)$ is sufficiently high. Second, the distance $dist(p,s)$ has an exponentially decaying effect on the score, meaning that feature points nearer to p contribute higher scores.

4.2 Query Processing for SP, GP, BB, and BB*

We now examine the extensions of the SP, GP, BB, and BB* algorithms for top-k spatial preference queries defined by the influence score in (8).

4.2.1 Incremental Computation Technique

Observe that the upper bound of $\tau_c^{inf}(p)$ is 1. Therefore, (3) still holds for the influence score, and the incremental computation technique (see Section 3.2) can still be applied in SP, GP, and BB.

4.2.2 Exact Score Computation for a Single Object

For the SP algorithm, we elaborate how to compute the score $\tau_c^{inf}(p)$ (see (8)) of an object $p \in \mathcal{D}$. This is challenging because some feature $s \in \mathcal{F}_c$ outside the ϵ-range of p may contribute to the score. Unlike the computation of the range score, we can no longer use the ϵ-range to restrict the search space.

Given an object point p and an entry e from the feature tree of $\mathcal{F}_c$, we define the upper bound function

$$\omega^{inf}(e,p) = \omega(e) \cdot 2^{-\frac{mindist(p,e)}{\epsilon}}. \tag{10}$$

In case e is a leaf entry (i.e., a feature point s), we have $\omega^{inf}(s,p) = \omega(s) \cdot 2^{-\frac{dist(p,s)}{\epsilon}}$. The following lemma shows that the value $\omega^{inf}(e,p)$ is an upper bound of $\omega^{inf}(e',p)$ for any entry e' in the subtree of e.

Lemma 2. *Let e and e' be entries from the feature tree $\mathcal{F}_c$ such that e' is in the subtree of e. It holds that $\omega^{inf}(e,p) \geq \omega^{inf}(e',p)$, for any object point $p \in \mathcal{D}$.*

Proof. Let p be any object point $p \in \mathcal{D}$. Since e' falls into the subtree of e, we have: $mindist(p,e) \leq mindist(p,e')$. As $\mathcal{F}_c$ is a `MAX` aR-tree, we have: $\omega(e) \geq \omega(e')$. Thus, we have: $\omega^{inf}(e,p) \geq \omega^{inf}(e',p)$. □

As an example, Fig. 6b shows an object point p and three entries e_1, e_2, e_3 from the same feature tree. Note that e_2 and e_3 are in the subtree of e_1. The dotted lines indicate the minimum distance from p to e_1 and e_3, respectively. Thus, we have $\omega^{inf}(e_1,p) = 0.8 \cdot 2^{-\frac{0.2}{0.2}} = 0.4$ and $\omega^{inf}(e_3,p) = 0.7 \cdot 2^{-\frac{0.3}{0.2}} = 0.247$. Clearly, $\omega^{inf}(e_1,p)$ is larger than $\omega^{inf}(e_3,p)$.

By using Lemma 2, we apply the best-first approach to compute the exact component score $\tau_c(p)$ for the point p; Algorithm 6 employs a max-heap H in order to visit the entries of the tree in descending order of their ω^{inf} values. We first insert the root of the tree $\mathcal{F}_c$ into H, and initialize $\tau_c(p)$ to 0. The loop at Line 4 continues as long as H is not empty. At Line 5, we deheap an entry e from H. If the value $\omega^{inf}(e,p)$ is above the current $\tau_c(p)$, then there is potential to update $\tau_c(p)$ by using some point in the subtree of e. In that case, we read the child node pointed to by e, and examine each entry e' in that node (Lines 7-8). If e' is a nonleaf entry, it is inserted into H provided that its $\omega^{inf}(e',p)$ value is above $\tau_c(p)$. Otherwise, it is used to update $\tau_c(p)$.

Algorithm 6. Object Influence Score Algorithm

```
    algorithm Object_Influence(Point $p$, Value $c$, Value $\epsilon$)
1:  $H :=$ new max-heap (with $\omega^{inf}$ value as key);
2:  insert $\langle \mathcal{F}_c.root, 1.0 \rangle$ into $H$;
3:  $\tau_c(p) := 0$;
4:  while $H$ is not empty do
5:      deheap an entry $e$ from $H$;
6:      if $\omega^{inf}(e,p) > \tau_c(p)$ then
7:          read the child node $CN$ pointed to by $e$;
8:          for each entry $e'$ of $CN$ do
9:              if $CN$ is a nonleaf node then
10:                 if $\omega^{inf}(e',p) > \tau_c(p)$then
11:                     insert $\langle e', \omega^{inf}(e',p) \rangle$ into $H$;
12:             else                    ▷ update component score
13:                 $\tau_c(p) := \max\{\tau_c(p), \omega^{inf}(e',p)\}$;
```

4.2.3 Group Computation and Upper Bound Computation

Recall that, for the case of range scores, both the GP and BB algorithms apply the group computation technique (Algorithm 2) for concurrently computing the component score $\tau_c(p)$ for every object point p in a given set V. Now, Algorithm 6 can be modified as follows to support concurrent computation of influence scores. First, the parameter p is replaced by a set V of objects. Second, we initialize the value $\tau_c(p)$ for each object $p \in V$ at Line 3 and perform the score update for each $p \in V$ at Line 13. Third, the conditions at Lines 6 and 10 are checked whether they are satisfied by some object $p \in V$.

In addition, the BB algorithm (see Algorithm 3) needs to compute the upper bound component score $\mathcal{T}_c(e)$ for all nonleaf entries in the current node simultaneously. Again, Algorithm 6 can be modified for this purpose.

4.2.4 Optimized Computation of Scores in BB*

Given an entry e (from a feature tree), we define the upper bound score of e using a set V of points as

$$\omega^{inf}(e,V) = \max_{p \in V} \omega^{inf}(e,p). \tag{11}$$

The BB* algorithm applies Algorithm 4 to compute the range scores for a set V of object points. With (11), we can modify Algorithm 4 to compute the influence score, with the following changes. First, the heap H_c (at Line 2) is used to organize its entries e in descending order of the key $\omega^{inf}(e,V)$, and the value $\omega(e)$ (at Line 10) is replaced by $\omega^{inf}(e,V)$. Second, the restrictions based on the ϵ-range (at Lines 11-12, 19, and 22) are removed. Third, the value $\omega(e')$ (at Line 23) needs to be replaced by $\omega^{inf}(e',p)$.

4.3 Query Processing for FJ

The FJ algorithm can be adapted for the influence score, but with two changes. Recall that the tuple $\langle f_1, f_2, \ldots, f_m \rangle$ is said to be a combination if f_c is an entry in the feature tree $\mathcal{F}_c$, for any $c \in [1,m]$.

First, (6) 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 so far γ.

Second, (5) is now a loose upper bound value for the influence score because it ignores the distances among the

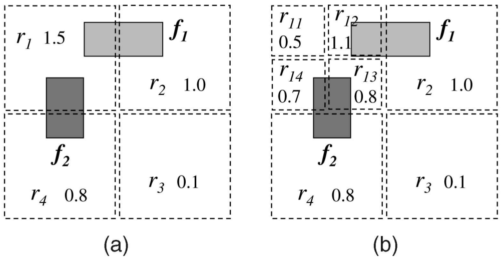


Fig. 7. Deriving upper bound of the influence score for FJ. (a) First iteration. (b) Second iteration.

entries f_c. Therefore, we need to develop a tighter upper bound for the influence score.

The following lemma shows that, given a set Φ of rectangles that partition the spatial domain $\mathcal{DOM}$, the value $\max_{r \in \Phi} \sum_{c=1}^{m} \omega^{inf}(f_c, r)$ is an upper bound of the value $\sum_{c=1}^{m} \omega^{inf}(f_c, p)$ for any point p (in $\mathcal{DOM}$).

Lemma 3. *Let Φ be a set of rectangles which partition the spatial domain $\mathcal{DOM}$. Given the tree entries $f_1, f_2, \ldots, f_m$ (from the respective feature trees), it holds that $\max_{r \in \Phi} \sum_{c=1}^{m} \omega^{inf}(f_c, r) \geq \sum_{c=1}^{m} \omega^{inf}(f_c, p)$, for any point $p \in \mathcal{DOM}$.*

Proof. Let p be a point in $\mathcal{DOM}$. There exists an rectangle $r' \in \Phi$ such that p falls into r'. Thus, for any $c \in [1, m]$, we have $mindist(f_c, r') \leq mindist(f_c, p)$, and derive $\omega^{inf}(f_c, r') \geq \omega^{inf}(f_c, p)$. By summing all components, we obtain $\sum_{c=1}^{m} \omega^{inf}(f_c, r') \geq \sum_{c=1}^{m} \omega^{inf}(f_c, p)$. As $r' \in \Phi$, we also have $\max_{r \in \Phi} \sum_{c=1}^{m} \omega^{inf}(f_c, r) \geq \sum_{c=1}^{m} \omega^{inf}(f_c, r')$. Therefore, the lemma is proved. □

In fact, the above upper bound value

$$\max_{r \in \Phi} \sum_{c=1}^{m} \omega^{inf}(f_c, r)$$

can be tightened by dividing the rectangles of Φ into smaller rectangles.

Fig. 7a shows the combination $\langle f_1, f_2 \rangle$, whose entries belong to the feature trees $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. We first partition the domain space into four rectangles r_1, r_2, r_3, r_4, and then compute their upper bound values (shown in the figure). Thus, the current upper bound score of $\langle f_1, f_2 \rangle$ is taken as: $\max\{1.5, 1.0, 0.1, 0.8\} = 1.5$. To tighten the upper bound, we pick the rectangle (r_1) with the highest value and partition it into four rectangles $r_{11}, r_{12}, r_{13}, r_{14}$ (see Fig. 7b). Now, the upper bound score of $\langle f_1, f_2 \rangle$ becomes: $\max\{0.5, 1.1, 0.8, 0.7, 1.0, 0.1, 0.8\} = 1.1$. By applying this method iteratively, the upper bound score can be gradually tightened.

Algorithm 7 is the pseudocode for computing the upper bound score for the combination $\langle f_1, f_2, \ldots, f_m \rangle$ of feature entries. The parameter γ represents the best score found so far (in FJ). The value I_{max} is used to control the number of iterations in the algorithm; its typical value is 20. At Line 1, we employ a max-heap H to organize its rectangles in descending order of their upper bound scores. Then, we insert the spatial domain rectangle into H. The loop at Line 4 continues while H is not empty and $I_{max} > 0$. After deheaping a rectangle r from H (Line 5), we partition it into four child rectangles. Each child rectangle r' is inserted into H if its upper bound score is above γ. We then decrement I_{max} (at Line 10). At the end (Lines 11-14), if the heap H is not empty, then algorithm returns the key value of H's top entry as the upper bound score. Such a value is guaranteed to be the maximum upper bound value in the heap. Otherwise (i.e., empty H), the algorithm returns γ as the upper bound score because all rectangles with score below γ have been pruned.

Algorithm 7. FJ Upper Bound Computation Algorithm

```
algorithm FJ_Influence(Value $\epsilon$, Value $\gamma$, Value $I_{max}$,
    Entries $f_1, f_2, \ldots, f_m$)
 1: $H :=$ new max-heap (with upper bound score as key);
 2: let $\mathcal{DOM}$ be the rectangle of the spatial domain;
 3: insert $\langle \mathcal{DOM}, \sum_{c=1}^{m} \omega(f_c) \rangle$ into $H$;
 4: while $I_{max} > 0$ and $H$ is not empty do
 5:     deheap a rectangle $r$ from $H$;
 6:     partition $r$ into four child rectangles;
 7:     for each child rectangle $r'$ of $r$ do
 8:         if $\sum_{c=1}^{m} \omega^{inf}(f_c, r') > \gamma$ then
 9:             insert $\langle r', \sum_{c=1}^{m} \omega^{inf}(f_c, r') \rangle$ into $H$;
10:     $I_{max} := I_{max} - 1$;
11: if $H$ is not empty then
12:     return the key value of the top entry of $H$;
13: else
14:     return $\gamma$;
```

4.4 Extension to Generic Influence Scores

Our algorithms can be also applied to other types of influence score functions. Given a function $\mathcal{U} : \Re \rightarrow \Re$, we model a score function as $\omega^{inf}(s, p) = \omega(s) \cdot \mathcal{U}(dist(p, s))$, where p is an object point and $s \in \mathcal{F}_c$ is a feature point.

Let e be a feature tree entry. The crux of our solution is to redefine the upper bound function $\omega^{inf}(e, p)$ (like in (10)) such that $\omega^{inf}(e, p) \geq \omega^{inf}(s, p)$, for any feature point s in the subtree of e.

In fact, the upper bound function can be expressed as $\omega^{inf}(e, p) = \omega(e) \cdot \mathcal{U}(d)$, where d is a distance value. Observe that d must fall in the interval $[mindist(p, e), maxdist(p, e)]$. Thus, we apply a numerical method (e.g., the bisection method) to find the value $d \in [mindist(p, e), maxdist(p, e)]$ that maximizes the value of $\mathcal{U}$.

For the special case that $\mathcal{U}$ is a monotonic decreasing function, we can simply set $d = mindist(p, e)$ because it definitely maximizes the value of $\mathcal{U}$.

5 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. Section 5.1 describes the experimental settings. Sections 5.2 and 5.3 study the performance of the proposed algorithms for queries with range scores and influence scores, respectively. We then present our experimental findings on real data in Section 5.4.

TABLE 2
Range of Parameter Values

Parameter	Values
Aggregate function	**SUM**, MIN, MAX
Buffer size (%)	0.1, 0.2, **0.5**, 1, 2, 5, 10
Object data size, $\|\mathcal{D}\|$ ($\times 1000$)	100, **200**, 400, 800, 1600
Feature data size, $\|\mathcal{F}\|$ ($\times 1000$)	50, **100**, 200, 400, 800
Number of results, k	**1**, 2, 4, 8, 16, 32, 64
Number of features, m	1, **2**, 3, 4, 5
Query range, ϵ	10, 20, **50**, 100, 200

5.1 Experimental Settings

We used both real and synthetic data for the experiments. The real data sets will be described in Section 5.4. 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 100K points. The point coordinates of all data sets are normalized to the 2D space $[0, 10000]^2$.

For a feature data set $\mathcal{F}_c$, we generated qualities for its points such that they simulate a real-world scenario: facilities close to (far from) a town center often have high (low) quality. For this, a single *anchor* point $s_\star$ 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 $\mathcal{F}_c$ from the anchor $s_\star$. Then, the quality of a feature point s is generated as

$$\omega(s) = \left(\frac{dist_{max} - dist(s, s_\star)}{dist_{max} - dist_{min}}\right)^{\vartheta}, \qquad (12)$$

where $dist(s, s_\star)$ stands for the distance between s and $s_\star$, and ϑ controls the skewness (default: 1.0) of quality distribution. In this way, the qualities of points in $\mathcal{F}_c$ lie in $[0, 1]$ and the points closer to the anchor have higher qualities. Also, the quality distribution is highly skewed for large values of ϑ.

We study the performance of our algorithms with respect to various parameters, which are displayed in Table 2 (their default values are shown in bold). In each experiment, only one parameter varies while the others are fixed to their default values.

5.2 Performance on Queries with Range Scores

This section studies the performance of our algorithms for top-k spatial preference queries on range scores.

Table 3 shows the I/O cost and execution time of the algorithms, for different aggregate functions (SUM, MIN, and MAX). GP has lower cost than SP because GP computes the scores of points within the same leaf node concurrently. The incremental computation technique (used by SP and GP) 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 (see Section 3.6). This explains the performance of SP and GP across different aggregate functions. However, the cost of the other methods are mainly influenced by the effectiveness of pruning. BB employs an effective technique to prune unqualified nonleaf entries in the object tree so it outperforms GP. The optimized score computation method enables BB* to save on average 20 percent I/O and 30 percent time of BB. FJ outperforms its competitors as it discovers qualified combination of feature entries early.

TABLE 3
Effect of the Aggregate Function, Range Scores

SUM function	SP	GP	BB	BB*	FJ
I/O	350927	22594	2033	1535	489
Time (s)	635.0	32.7	3.0	2.0	1.3

MIN function	SP	GP	BB	BB*	FJ
I/O	235602	16254	611	615	47
Time (s)	426.8	22.7	0.9	0.8	0.2

MAX function	SP	GP	BB	BB*	FJ
I/O	402704	26128	228	186	8
Time (s)	742.8	38.2	0.3	0.2	0.1

We ignore SP in subsequent experiments, and compare the cost of the remaining algorithms on synthetic data sets with respect to different parameters.

Next, we empirically justify the choice of using level-1 entries of feature trees $\mathcal{F}_c$ for the upper bound score computation routine in the BB algorithm (see Section 3.3). In this experiment, we use the default parameter setting and study how the number of node accesses of BB is affected by the level of $\mathcal{F}_c$ used. Table 4 shows the decomposition of node accesses over the tree $\mathcal{D}$ and the trees $\mathcal{F}_c$, and the statistics of upper bound score computation. Each accessed nonleaf node of $\mathcal{D}$ invokes a call of the upper bound score computation routine.

When level-0 entries of $\mathcal{F}_c$ are used, each upper bound computation call incurs a high number (617.5) of node accesses (of $\mathcal{F}_c$). 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 $\mathcal{D}$. Observe that the total cost is minimized when level-1 entries (of $\mathcal{F}_c$) 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 $\mathcal{D}$.

Fig. 8 plots the cost of the algorithms as a function of the buffer size. As the buffer size increases, the I/O of all algorithms drops. FJ remains the best method, BB* the second, and BB the third; all of them outperform GP by a wide margin. Since the buffer size does not affect the pruning effectiveness of the algorithms, it has a small impact on the execution time.

Fig. 9 compares the cost of the algorithms with respect to the object data size $|\mathcal{D}|$. Since the cost of FJ is dominated by the cost of joining feature data sets, it is insensitive to $|\mathcal{D}|$. On the other hand, the cost of the other methods (GP, BB, and BB*) increases with $|\mathcal{D}|$, as score computations need to be done for more objects in $\mathcal{D}$.

TABLE 4
Effect of the Level of $\mathcal{F}_c$ Used for Upper Bound Score Computation in the BB Algorithm

Level	Node accesses (NA)			Upper bound score computation	
	Total	of $\mathcal{D}$	of $\mathcal{F}_c$	# of calls	NA of $\mathcal{F}_c$ per call
0	3350	53	3297	4	617.5
1	2365	130	2235	4	15
2	13666	930	12736	14	2

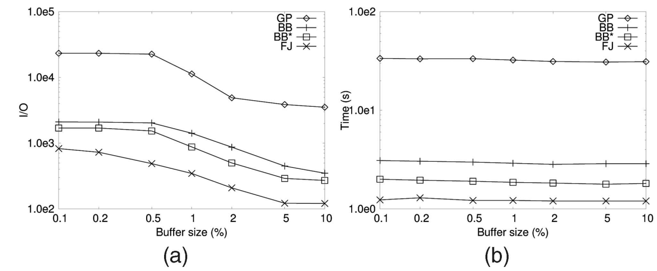


Fig. 8. Effect of buffer size, range scores. (a) I/O. (b) Time.

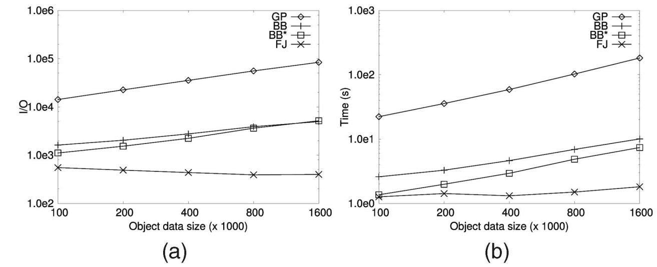


Fig. 9. Effect of $|\mathcal{D}|$, range scores. (a) I/O. (b) Time.

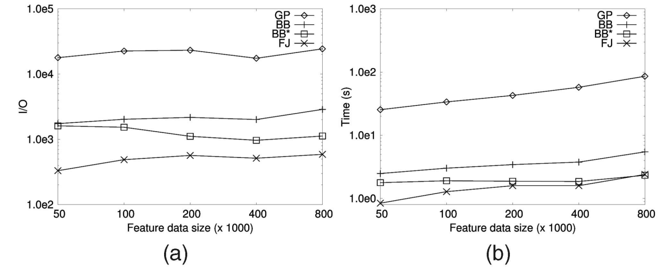


Fig. 10. Effect of $|\mathcal{F}|$, range scores. (a) I/O. (b) Time.

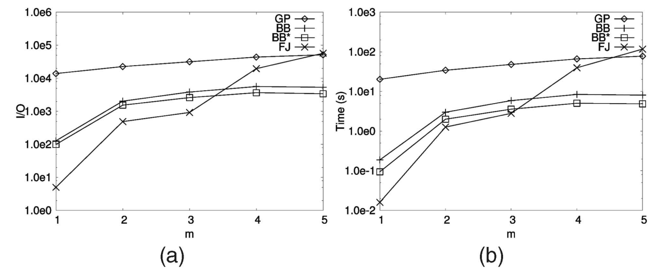


Fig. 11. Effect of m, range scores. (a) I/O. (b) Time.

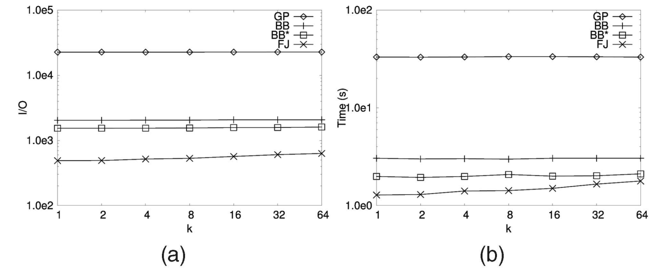


Fig. 12. Effect of k, range scores. (a) I/O. (b) Time.

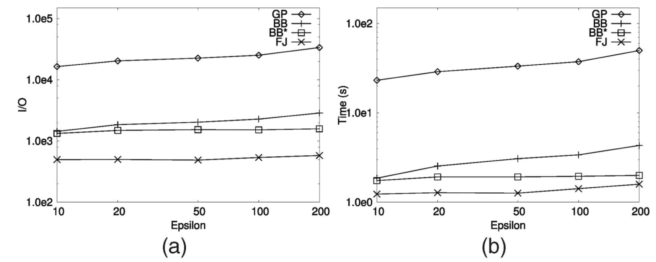


Fig. 13. Effect of ϵ, range scores. (a) I/O. (b) Time.

Fig. 10 plots the I/O cost of the algorithms with respect to the feature data size $|\mathcal{F}|$ (of each feature data set). As $|\mathcal{F}|$ increases, the cost of GP, BB, and FJ increases. In contrast, BB* experiences a slight cost reduction as its optimized score computation method (for objects and nonleaf entries) is able to perform pruning early at a large $|\mathcal{F}|$ value.

Fig. 11 plots the cost of the algorithms with respect to the number m of feature data sets. The costs of GP, BB, and BB* rise linearly as m increases because the number of component score computations is at most linear to m. On the other hand, the cost of FJ increases significantly with m, because the number of qualified combinations of entries is exponential to m.

Fig. 12 shows the cost of the algorithms as a function of the number k of requested results. GP, BB, and BB* compute the scores of objects in $\mathcal{D}$ in batches, so their performance is insensitive to k. As k increases, FJ has weaker pruning power and its cost increases slightly.

Fig. 13 shows the cost of the algorithms, when varying the query range ϵ. As ϵ increases, all methods access more nodes in feature trees to compute the scores of the points. The difference in execution time between BB* and FJ shrinks as ϵ increases. In summary, although FJ is the clear winner in most of the experimental instances, its performance is significantly affected by the number m of feature data sets. BB* is the most robust algorithm to parameter changes and it is recommended for problems with large m.

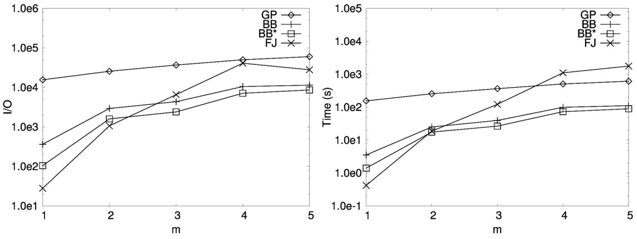
5.3 Performance on Queries with Influence Scores

We proceed to examine the cost of our algorithms for top-k spatial preference queries on influence scores.

Fig. 14 compares the cost of the algorithms with respect to the number m of feature data sets. The cost follows the trend in Fig. 11. Again, the number of combinations examined by FJ increases exponentially with m so its cost increases rapidly.

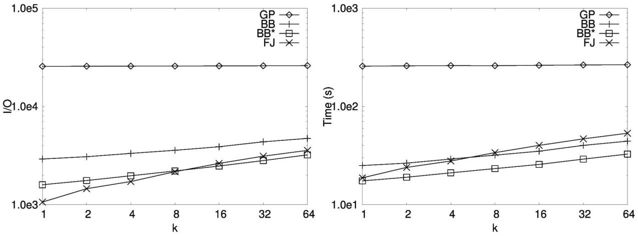
Fig. 15 plots the cost of the algorithms by varying the number k of requested results. Observe that FJ becomes more expensive than BB* (in both I/O and time) when the value of k is beyond 8. This is attributed to two reasons. First, FJ incurs extra computational cost as it needs to invoke Algorithm 7 for computing the upper bound score of a combination of feature entries. Second, FJ incurs high I/O cost to identify objects in $\mathcal{D}$ that produce high scores with the current combination of features.

Fig. 16 shows the cost of the algorithms as a function of the parameter ϵ. Interestingly, the trend here is different from the one in Fig. 13. According to (8), when ϵ decreases, the influence score also decreases, rendering it more difficult to distinguish the scores among different objects. Thus, the cost of BB, BB*, and FJ becomes high at a low ϵ value. Summing up, for the newly introduced influence score, FJ is more sensitive to parameter changes and it loses to BB* not only when there are multiple feature data sets, but also at large k.



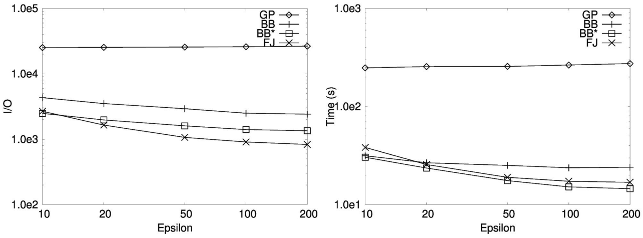
(a) (b)

Fig. 14. Effect of m, influence scores. (a) I/O. (b) Time.



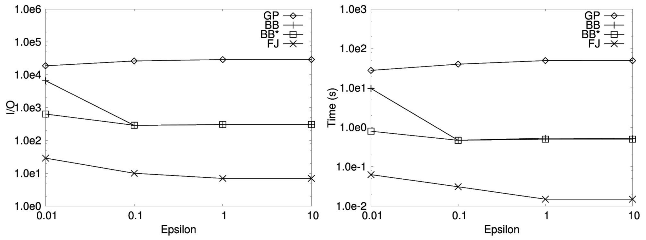
(a) (b)

Fig. 15. Effect of k, influence scores. (a) I/O. (b) Time.



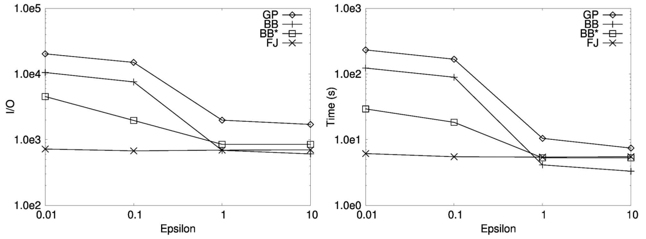
(a) (b)

Fig. 16. Effect of ϵ, influence scores. (a) I/O. (b) Time.



(a) (b)

Fig. 17. Effect of ϵ, range scores, real data. (a) I/O. (b) Time.



(a) (b)

Fig. 18. Effect of ϵ, influence scores, real data. (a) I/O. (b) Time.

5.4 Results on Real Data

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 website, 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. Each remaining location is normalized to a point in the 2D space $[0, 10,000]^2$. One data set is used as the object data set and the other two are used as feature data sets. The object data set $\mathcal{D}$ contains 11,399 camping locations. The feature data set $\mathcal{F}_1$ contains 30,921 hotel records, each with a room price (quality) and a location. The feature data set $\mathcal{F}_2$ has 3,848 records of Wal-Mart stores, each with a gasoline availability (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).

Fig. 17 plots the cost of the algorithms with respect to ϵ, for queries with range scores. At a very small ϵ value, most of the objects have the zero score as they have no feature points within their neighborhood. This forces BB, BB*, and FJ to access a larger number of objects (or feature combinations) before finding an object with nonzero score, which can then be used for pruning other unqualified objects.

Fig. 18 compares the cost of the algorithms with respect to ϵ, for queries with influence scores. In general, the cost follows the trend in Fig. 16. BB* outperforms BB at low ϵ value whereas BB incurs a slightly lower cost than BB* at a high ϵ value. Observe that the cost of BB and BB* is close to that of FJ when ϵ is sufficiently high. In summary, the relative performance between the algorithms in all experiments is consistent to the results on synthetic data.

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 five 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 nonleaf 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 nonleaf entries). The algorithm FJ 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 our experimental findings, BB* is scalable to large data sets and it is the most robust algorithm with respect to various parameters. However, FJ is the best algorithm in cases where the number m of feature data sets is low and each feature data set is small.

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.

ACKNOWLEDGMENTS

This work was supported by grant HKU 715509E from Hong Kong RGC.

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] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," *Proc. Int'l Symp. Principles of Database Systems (PODS),* 2001.

[8] I.F. Ilyas, W.G. Aref, and A. Elmagarmid, "Supporting Top-k Join Queries in Relational Databases," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB),* 2003.

[9] N. Mamoulis, M.L. Yiu, K.H. Cheng, and D.W. Cheung, "Efficient Top-k Aggregation of Ranked Inputs," *ACM Trans. Database Systems,* vol. 32, no. 3, p. 19, 2007.

[10] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, "Efficient OLAP Operations in Spatial Data Warehouses," *Proc. Int'l Symp. Spatial and Temporal Databases (SSTD),* 2001.

[11] S. Hong, B. Moon, and S. Lee, "Efficient Execution of Range Top-k Queries in Aggregate R-Trees," *IEICE Trans. Information and Systems,* vol. 88-D, no. 11, pp. 2544-2554, 2005.

[12] T. Xia, D. Zhang, E. Kanoulas, and Y. Du, "On Computing Top-t Most Influential Spatial Sites," *Proc. 31st Int'l Conf. Very Large Data Bases (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 of The Min-Dist Optimal-Location Query," *Proc. 32nd Int'l Conf. Very Large 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 Reverse Proximity Query Problems," *Proc. 16th ACM Int'l Conf. Advances in Geographic Information Systems (GIS),* 2008.

[17] M.L. Yiu, P. Karras, and N. Mamoulis, "Ring-Constrained Join: Deriving Fair Middleman Locations from Pointsets via a Geometric 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'l Conf. Data Eng. (ICDE),* 2008.

[19] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient Query Processing in Geographic Web Search Engines," *Proc. ACM SIGMOD,* 2006.

[20] V.S. Sengar, T. Joshi, J. Joy, S. Prakash, and K. Toyama, "Robust Location Search from Text Queries," *Proc. 15th Ann. ACM Int'l Symp. Advances in Geographic Information Systems (GIS),* 2007.

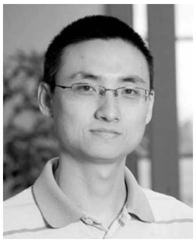
[21] S. Berchtold, C. Boehm, D. Keim, and H. Kriegel, "A Cost Model for Nearest Neighbor Search in High-Dimensional Data Space," *Proc. ACM Symp. Principles of Database Systems (PODS),* 1997.

[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," *ACM Trans. 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.

Man Lung Yiu received the bachelor's degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an assistant professor in the Department of Computing, Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular, query processing topics on spatiotemporal data and multidimensional data.



Hua Lu received the BSc and MSc degrees from Peking University, China, in 1998 and 2001, respectively, and the PhD degree in computer science from the National University of Singapore in 2007. He is currently an assistant professor in the Department of Computer Science, Aalborg University, Denmark. His research interests include skyline queries, spatiotemporal databases, geographic information systems, and mobile computing. He is a member of the IEEE.

Nikos Mamoulis received the diploma in computer engineering and informatics from the University of Patras, Greece, in 1995, and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2000. He is currently an associate professor in the Department of Computer Science, University of Hong Kong, which he joined in 2001. In the past, he has worked as a research and development engineer at the Computer Technology Institute, Patras, Greece and as a postdoctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), the Netherlands. During 2008-2009, he was on leave to the Max-Planck Institute for Informatics (MPII), Germany. His research focuses on the management and mining of complex data types, including spatial, spatiotemporal, object-relational, multimedia, text, and semistructured data. He has served on the program committees of more than 70 international conferences and workshops on data management and data mining. He was the general chair of SSDBM 2008, the PC chair of SSTD 2009, and he organized the SSTDM 2006 and DBRank 2009 workshops. He has served as PC vicechair of ICDM 2007, ICDM 2008, and CIKM 2009. He was the publicity chair of ICDE 2009. He is an editorial board member for *Geoinformatica Journal* and was a field editor of the *Encyclopedia of Geographic Information Systems*.

Michail Vaitis received the diploma in 1992 and the PhD degree in computer engineering and informatics from the University of Patras, Greece, in 2001. He was collaborating for five years with the Computer Technology Institute (CTI), Greece, as a research and development engineer, working on hypertext and database systems. Now, he is an assistant professor at the Department of Geography, University of the Aegean, Greece, which he joined in 2003. His research interests include geographical databases, spatial data infrastructures, and hypermedia models and services.