Efficient Top-k Spatial Distance Joins*

Shuyao Qi[1], Panagiotis Bouros[2,**], and Nikos Mamoulis[1]

[1] Department of Computer Science
The University of Hong Kong
{syqi2,nikos}@cs.hku.hk
[2] Department of Computer Science
Humboldt-Universität zu Berlin, Germany
bourospa@informatik.hu-berlin.de

Abstract. Consider two sets of spatial objects R and S, where each object is assigned a score (e.g., ranking). Given a spatial distance threshold ϵ and an integer k, the top-k spatial distance join (k-SDJ) returns the k pairs of objects, which have the highest combined score (based on an aggregate function γ) among all object pairs in $R \times S$ which have spatial distance at most ϵ. Despite the practical application value of this query, it has not received adequate attention in the past. In this paper, we fill this gap by proposing methods that utilize both location and score information from the objects, enabling top-k join computation by accessing a limited number of objects. Extensive experiments demonstrate that a technique which accesses blocks of data from R and S ordered by the object scores and then joins them using an aR-tree based module performs best in practice and outperforms alternative solutions by a wide margin.

1 Introduction

The spatial join operator retrieves pairs of objects that satisfy a spatial predicate. Spatial joins have been extensively studied [3,6,17,7,15] due to their applicability and potentially high execution cost. Still, this query type only focuses on spatial attributes, while in many applications spatial objects have additional attributes. For instance, restaurants shown in websites like Foursquare and Yelp are assigned user-generated ratings. As another example, consider collections of spatial objects created in the context of emerging scientific fields like atmospheric, oceanographic, and environmental sciences with an expertise that ranges, from the modeling of global climatic change to the analysis of earth's tectonics. The objects in such collections are associated with measurements of several attributes varying from temperature and pressure to earth's gravity and seismic activity.

These attributes can be used to derive a ranking for the objects. Ranking has been considered by the database community in the context of top-k queries [5,9] and top-k joins [13,8,12,16] where ranked inputs are joined to derive objects or tuple pairs which maximize an aggregate function on scoring attributes. Consider, for example, the following top-k join query expressed in SQL:

* Work supported by grant HKU 714212E from Hong Kong RGC.
** This work was conducted while the author was with the University of Hong Kong.

M.A. Nascimento et al. (Eds.): SSTD 2013, LNCS 8098, pp. 1–18, 2013.

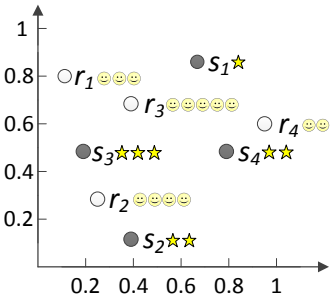


Fig. 1. Example of a top-k spatial distance join

```
SELECT R.id, S.id
FROM R, S
WHERE R.att = S.att
ORDER BY R.score + S.score DESC
STOP AFTER k;
```

The result of this query is the k pairs of objects (r, s) with $r \in R$ and $s \in S$ that qualify the equality predicate on their common attribute `att`, having the highest SUM of their `score` attributes.

Despite the vast availability of spatial objects associated with scoring attributes, to our knowledge, there exists no join operator that considers both spatial and score attributes at the same time.[1] On an attempt to fill this gap, we introduce the *top-k spatial distance join* (k-SDJ) query. Given two collections of spatial objects R and S that also carry a score attribute, the k-SDJ query retrieves a k-subset J of $R \times S$ such that for every pair of objects $(r, s) \in J$, r is spatially close to s based on a *distance threshold* ϵ (i.e., $dist(r, s) \leq \epsilon$, where $dist$ denotes the distance between the spatial locations of r and s), and for every $(r', s') \in R \times S - J$ such that $dist(r', s') \leq \epsilon$, it holds $\gamma(r, s) \geq \gamma(r', s')$, where γ is a *monotone aggregate function* (e.g., SUM) which combines the scores of two objects. k-SDJ finds application in tasks like recommending to the visitors of a city the k best pairs of restaurants and hotels within short distance that have the top combined ratings, or investigating the correlation between scientific attributes, e.g., identifying locations where earthquakes of high magnitude take place on a very large depth. For instance, Figure 1 illustrates a set R of four restaurants and a set S of four hotels. The objects carry a score shown next to every point. Assuming that the qualifying pairs should have Euclidean distance at most $\epsilon = 0.3$ and $\gamma = SUM$, the result of 2-SDJ contains pairs (r_2, s_3) with aggregate score 7 and (r_2, s_2) with aggregate score 6. Notice that, although $dist(r_4, s_4) < \epsilon$, pair (r_4, s_4) is not included in the query result because $\gamma(r_4, s_4) < \gamma(r_2, s_2) < \gamma(r_2, s_3)$. Further, while being the restaurant with the highest score, r_3 is not included in any result pair, as there is no hotel at a

[1] An exception is the work of [11] which, however, is restricted to a specific type of attributes (probabilities) and a specific aggregation function (product).

distance to r_3 smaller than 0.3. Note that k-SDJ is very similar to the top-k join problem in relational databases (see the example SQL query above); the only difference is that in k-SDJ the equality join predicate is replaced by a distance bounding predicate between the spatial locations of objects in R and S.

Contributions. In this paper, we study the efficient evaluation of the k-SDJ query. In brief, the key contributions of our work are summarized as follows:

- We introduce k-SDJ over two collections of spatial objects with scoring attributes. The k-SDJ query can be used either as a standalone operator or participate in complex query evaluation plans. For this purpose, we assume that the input collections are not indexed in advance.
- We present three algorithms, which access and process the data in different order; (i) the *Score-First algorithm* (SFA), which accesses the objects from R and S in decreasing order of their scores, (ii) the *Distance-First Algorithm*, which gives higher priority to the spatial distance join component of the query, and (iii) the *Block-based Algorithm* (BA), which performs block-wise evaluation, combining the benefits of SFA and DFA, without sharing their disadvantages. All techniques employ aR-trees [14] (albeit in different fashions) in order to combine spatial search with score-based pruning.
- We conduct extensive experiments to verify the effectiveness and efficiency of our proposed methods.

Outline. The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 presents algorithms for k-SDJ evaluation. Comprehensive experiments and our findings are reported in Section 4. Finally, Section 5 concludes the paper and discusses directions for future work.

2 Related Work

Our work is related to spatial joins, top-k queries and top-k joins, and spatial top-k joins. Sections 2.1 to 2.4 summarize related work done in these areas.

2.1 Spatial Joins

There exist two types of spatial distance join queries: the ϵ-*distance* and the k-*closest pairs* join. Given two collections of spatial objects R and S the ϵ-distance join identifies the object pairs (r, s) with $r \in R$, $s \in S$, such that $dist(r, s) \leq \epsilon$. An ϵ-distance join can be processed similarly to a spatial intersection join [2]. Specifically, assuming that each of the R and S collections are indexed by an R-tree, the two R-trees are concurrently traversed by recursively visiting pairs of entries (e_R, e_S) for which their MBRs have minimum distance at most ϵ. Minimizing the cost of computing the distance between an MBR and an object was studied in [3]. For non-indexed inputs, alternative spatial join algorithms can be applied (e.g., the algorithm of [1] based on external sorting and plane sweep). The k-closest pairs join computes, from two collections R and S, the k object

pairs (r, s), $r \in R$, $s \in S$, with the minimum spatial distance $dist(r, s)$. Two different approaches exist for k-closest pairs. In the incremental approach [6,17] the results are reported one-by-one in ascending order of their spatial distance. For non-incremental computation of closest pairs, [4] extends the nearest neighbor algorithm of [15] achieving in this way, minimum memory requirements and better access locality for tree nodes.

2.2 Top-k Queries

Fagin et al. [5] present an analytical study of various methods for top-k aggregation of ranked inputs by monotone aggregate functions. Consider a set of objects (e.g., restaurants) which have scores (i.e., rankings) at two or more different sources (e.g., different ranking websites). Given an aggregate function γ (e.g., SUM) the top-k query returns the k restaurants with the highest aggregated scores (from the different sources). Each source is assumed to provide a sorted list of the objects according to their atomic scores there; requests for random accesses of scores based on object identifiers may also be possible. For the case where both sorted and random accesses are possible, a *threshold algorithm* (TA) retrieves objects from the ranked inputs (e.g., in a round-robin fashion) and a priority queue is used to organize the best k objects seen so far. Let l_i be the last score seen in source S_i; $T = \gamma(l_1, ..., l_m)$ defines a lower bound for the aggregate score of objects never seen in any S_i yet. If the kth highest aggregate score found so far is no less than T, the algorithm is guaranteed to have found the top-k objects and terminates. For the case where only sorted accesses are possible, [12] presents an optimized approach.

2.3 Top-k Joins

The top-k query is a special case of a top-k join query, which performs rank aggregation on top of relational join results; recall the SQL query example in the Introduction, where two tables R and S are joined (based on their common attribute `att`), but only the top-k join results according to the `score` attributes are required to be output.

Ilyas et al. [8] proposed a binary operator, called hash-based rank-join (HRJN) for top-k joins, which produces results incrementally and therefore can be used multiple times in an multi-way join evaluation plan. Assume that the tuples of R and S are accessed incrementally based on their values in the `score` attribute. HRJN accesses tuples from R (or S) and joins them using the join key `att` with the buffered tuples of S (or R), which have previously accessed (these tuples are buffered and indexed by a hash-table). Join results are organized in a priority queue based on their aggregate scores. Let l_R, h_R (l_S, h_S) be the lowest and highest scores seen in R (S) so far; all join results currently in the queue having aggregate scores larger than $T = \max\{\gamma(h_R, l_S), \gamma(l_R, h_S)\}$ are guaranteed to have higher aggregate score than any join result not found so far and therefore can be output (or pipelined to the operator than follows HRJN). A follow-up work [16] identifies optimal strategies for pulling tuples from the inputs in a

multi-way top-k join and joining them with the buffered results of other inputs. An early work on multi-way top-k join evaluation is done by Natsev et al. [13].

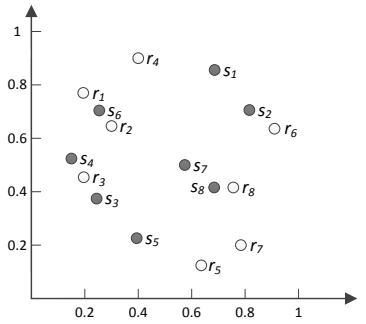
The binary top-k join operator (i.e., HRJN) can be adapted to solve k-SDJ; the only difference is that the equality join predicate in HRJN is replaced by a spatial distance predicate. In Section 3, we describe our *Score-First Algorithm* (SFA), which is based on this idea.

2.4 Spatial Top-k Joins

The term "top-k spatial join" is defined in [19]; however, this problem definition is very different from what we study in this paper; given two spatial datasets R and S, the query of [19] retrieves k objects in R intersecting the maximum number of objects from S. Therefore the ranking criterion based on the number of spatial intersections and not based on the aggregations of (non-spatial) scores from the two inputs. The only work to our knowledge, closely related to k-SDJ is [11], which studies a spatial join between two datasets R and S, containing spatial data associated with probabilistic values; in this case each object o (e.g., a biological cell) is defined by a set of probabilistic locations and it is also assigned a confidence p_o to belong to a specific cell class. Given two objects r and s from datasets R and S, respectively, a score of the (r, s) pair is defined by multiplying their confidence probabilities p_r and p_s, and also considering the distance $dist(r, s)$ between their uncertain locations. Then, the top-k probabilistic join between R and S returns the top-k object pairs in order of their scores. Compared to k-SDJ, the problem definition in [11] is different. The aggregate score function for k-SDJ does not involve the distance of the objects, but the distance is used in the join predicate. Further, the solution proposed in [11] is of limited applicability as it is bound to a specific aggregation function and can efficiently work only with L_1 distance.

3 Algorithms

In this section, we study evaluation techniques for k-SDJ. According to the query definition, the results are object pairs with (i) large aggregate scores and (ii) nearby spatial locations. First, we discuss two solutions, which extend work on top-k joins [13,8,12,16] and spatial joins [3,6,17,7,15], respectively, to prioritize either of the two query components; i.e., they either consider object scores or spatial distances first, respectively. We present these methods in Sections 3.1 and 3.2; they are optimized to employ aR-trees [14] (in a different fashion) in order to prune the search space during query evaluation. In Section 3.3, we present a framework which processes the objects in order of their scores, in a block-wise fashion and spatially joins the blocks, using bounds to early terminate accessing of blocks. Figure 2 illustrates the running example that we use to demonstrate our algorithms; two spatial datasets $R = \{r_1 \dots r_8\}$ and $S = \{s_1 \dots s_8\}$ with 8 points each. The point coordinates are shown on the left of the figure, while the two tables on the right show the objects in each collection in descending order of their scores. Table 1 shows the notation frequently used.



id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S

Fig. 2. Example of two datasets R and S with 8 points each

Table 1. Table of Symbols

Notation	Description
R/S	Collections with scored spatial objects
k	The number of required results
ϵ	The spatial distance threshold
γ	A monotone aggregate function
C	Candidate/result set of k-SDJ
θ	k-th smallest aggregate score in C

3.1 The Score-First Algorithm (SFA)

This method employs the framework of top-k join algorithms [8,16] to compute k-SDJ. In particular, a variant of the HRJN algorithm is applied using the spatial distance predicate instead of the equality predicate used in the original algorithm. The *Score-First Algorithm* (SFA) presumes that both R and S are ordered based on the object scores (e.g., as shown in Figure 2). This can be the case if they stem from underlying operators which produce such interesting orders; otherwise R and S need to be sorted before the application of SFA. SFA incrementally accesses objects either from R or from S. For each collection (e.g., R), it maintains an aR-tree [14] (e.g., A_R), which spatially organizes the buffered objects accessed so far.[2] In addition, SFA keeps track of the set C of distance join pairs found so far with the k highest aggregate scores and uses the lowest score θ in C as a bound for pruning and termination.

Algorithm 1 is a high-level pseudocode of SFA. After initializing C and the aR-trees, SFA incrementally accesses objects from R or S (we will shortly discuss about the access order). Assume that the current object is r accessed from R (i.e., $i = R$ and $o = r$ at Lines 5 and 7 of the pseudocode); the other case is symmetric. SFA performs the following steps:

[2] The aR-tree has identical structure and update algorithms as the R-tree, however, each non-leaf entry is augmented with the maximum score of all objects in the subtree pointed by it. Figure 3 illustrates the structure of two aR-trees for the data of Figure 2.

Algorithm 1. Score-First Algorithm (SFA)

Input: k, ϵ, γ, R, S
Output: C
1: initialize a min-heap C:=$\emptyset$ of candidate results; initialize θ:=$-\infty$
2: sort R and S based on score, if not already sorted
3: initialize aR-trees A_R:=$\emptyset$ and A_S:=$\emptyset$
4: **while** more objects exist in R and S **do**
5: i := next input to be accessed //either R or S
6: j := other input //either S or R
7: o := $get\_next(i)$ //get next object from input i
8: T:=max$\{\gamma(h_R, l_S), \gamma(l_R, h_S)\}$ //HRJN termination threshold
9: **while** (o' := $get\_next\_pair(o, A_j, \epsilon, \theta)$) $\neq$ $null$ and $T > \theta$ **do**
10: update C and θ using (o, o')
11: **if** $T \leq \theta$ **then**
12: **break** //result secured; no need to access more objects
13: insert o to A_i
14: **return** C

1. It updates T(Line 8)
2. It probes r against the aR-tree A_S for S to incrementally retrieve objects s from A_S such that $dist(r, s) \leq \epsilon$ and $\gamma(r, s) > \theta$ (function $get\_next\_pair()$ in Line 9 retrieves such objects in decreasing order of $\gamma(r, s)$);
3. For each qualifying pair (r, s) found, it updates C and θ (Line 10);
4. It checks whether the algorithm can terminate (Lines 11–12);
5. It inserts r to the aR-tree A_R for R (Line 13)

We now elaborate on the steps above. After each object access, SFA firstly updates the termination threshold $T = \max\{\gamma(h_R, l_S), \gamma(l_R, h_S)\}$ (Line 8). From the description of HRJN in Section 2.3, recall that l_R, h_R (l_S, h_S) are lowest and highest scores seen in R (S) so far (initially they are set as the maximum score in R (S)). In Step 2, aR-tree search on A_S is performed as a *score-based incremental ϵ-distance range query* centered at r (function $get\_next\_pair()$ in Line 9); during search, entries whose MBRs are further than ϵ from r are pruned and the remaining ones are prioritized based on their aggregate scores (i.e., the maximum score of any object indexed under them). Specifically, for each entry e, $\gamma(r, e.score)$ is computed (where $e.score$ is the aggregate score for e in the aR-tree) and if it is found not larger than θ, then the entry is pruned (as it would not be possible to find an object $s \in S$ in the subtree pointed by e, such that $\gamma(r, s) > \theta$). Otherwise, the entry is inserted in a *priority queue* which guides the aR-tree search to retrieve the spatial join pairs (r, s) in decreasing order of $s.score$ (note that this results in retrieving pairs in decreasing order of $\gamma(r, s)$, since r is fixed).

Whenever a new pair (r, s) is found, C and θ are updated immediately in order to tighten the θ bound and potentially prune additional aR-tree nodes during search: if $|C| < k$, (r, s) is inserted into C regardless of its aggregation score; otherwise, (r, s) is inserted into C only if $\gamma(r, s) > \theta$; in this case (r, s) *replaces* the k-th pair in C, such that C always keeps the best k pairs found so far. In either case, θ is updated to the k-th aggregate score in C. During the computation of new join results, as soon as $T \leq \theta$, SFA terminates reporting C as the k-SDJ result. Finally, r is inserted to the aR-tree A_R for R, which is used to probe objects from S.

We note that at Line 5 SFA chooses as input (R or S) to read the next object from the collection with the higher last seen score, according to the rationale of HRJN [8]; this increases the chances that the threshold T drops and that SFA terminates earlier.

To illustrate SFA, consider the example of Figure 2 and a k-SDJ query with $k = 1$, $\epsilon = 0.1$, and $\gamma = SUM$. After the first access (r_1 from R), there is no join result (since A_S is currently empty), so r_1 is just inserted to A_R. Then s_1 is accessed from S; s_1 is probed against A_R and no match is found (i.e., $dist(r_1, s_1) > \epsilon$). Then, since $l_R = 1.0 > l_S = 0.9$, r_2 is accessed and joined (unsuccessfully) with A_S; then s_2 and s_3 are accessed in turn, still without producing any distance join results. When r_3 is accessed, it is joined with A_S (now containing $\{s_1, s_2, s_3\}$) and produces the first join result (r_3, s_3), which is added to C; now $\theta = \gamma(r_3, s_3) = 1.6$. Note that T is currently $\max\{\gamma(1.0, 0.8), \gamma(0.8, 0.9)\} = 1.8 > \theta$, which means that a possibly better pair can be found and SFA cannot terminate yet. The next accessed object is s_4, which is joined with A_R, finding result (r_3, s_4), which is not better than the current top pair (r_3, s_3) (Note that in this case $get\_next\_pair(s_4, A_R, \epsilon, \theta)$ will not return r_3 but just $null$, because it uses θ for pruning). Next, both r_4 and s_5 give no new join pairs. Then s_6 is retrieved but still cannot produce a join pair with score better than θ; therefore SFA, after having updated T to 1.5, terminates reporting $C = \{(r_3, s_3)\}$.

SFA is expected to be fast, if the join results are found only after few accesses over the sorted collections R and S. If the best pairs include objects deep in the sorted collections, the overhead to maintain and probe the aR-trees can be high.

3.2 The Distance-First Algorithm (DFA)

The second evaluation technique extends a spatial distance join algorithm to compute the object pairs from R and S, which qualify the spatial threshold θ, *incrementally* in decreasing order of their aggregate scores. The algorithm terminates as soon as k pairs have been generated.

For implementing the spatial distance join, we could apply algorithms like the R-tree join [2], assuming that R and S are already indexed by R-trees, or methods that spatially join non-indexed inputs, like the (external memory) plane sweep algorithm [1], which first sorts R and S based on one of their coordinates and then sweeps a line along the sort axis to compute the results. The above approaches, however, do not have a way of prioritizing the join result computation according to the aggregate scores of qualifying distance join pairs.

We now present our optimized approach, which also employs aR-tree indices, however, it computes k-SDJ in a different way compared to SFA. The *Distance-First Algorithm* (DFA) (Algorithm 2) assumes that both R and S are indexed by two aR-trees A_R and A_S (if the trees do not exist, DFA bulk-loads them in a pre-processing phase, using the algorithm of [10]). DFA spatially joins the two trees by adapting the classic algorithm of [2] to traverse them not in a depth-first, but in a *best-first* order, which (1) still prunes entry pairs (e_R, e_S), $e_R \in A_R, e_S \in A_S$ for which $dist(e_R, e_S) > \epsilon$ ($dist$ here denotes the minimum distance between the MBRs of the two entries), but (2) prioritizes the entry pairs

Algorithm 2. Distance-First Algorithm (DFA)

Input: k, ϵ, γ, R, S
Output: k-SDJ results incrementally
1: build aR-trees A_R on R and A_S on S, if not already indexed
2: initialize a max-heap H_e of aR-tree entry pairs (e_R, e_S) organized by $\gamma(e_R, e_S)$
3: **for** each pair (e_R, e_S) in $A_R.root \times A_S.root$ **do**
4: **if** $dist(e_R, e_S) \leq \epsilon$ **then**
5: push (e_R, e_S) into H_e
6: **while** $H_e \neq \emptyset$ **do**
7: $(e_R, e_S) = H_e$.dequeue()
8: **if** e_R and e_S are non-leaf node entries **then**
9: $n_R :=$ node of A_R pointed by e_R; $n_S :=$ node of A_S pointed by e_S
10: **for** each entry $e'_R \in n_R$ and each entry $e'_S \in n_S$ **do**
11: **if** $dist(e'_R, e'_S) < \epsilon$ **then**
12: push (e'_R, e'_S) into H_e
13: **else**
14: output (e_R, e_S) as next k-SDJ result

to be examined based on $\gamma(e_R, e_S)$ (here, γ is applied on the aggregate scores stored at the entries). In other words, the entry pairs which have the maximum aggregate score are examined first during the join; this order guarantees that the qualifying object pairs will be computed incrementally in decreasing order of their aggregate scores. For this purpose, DFA initially puts in a priority queue (i.e., max heap) H_e all pairs of root entries within distance ϵ from each other in the two trees (Line 3 of Algorithm 2). Pairs of entries from H_e are de-heaped in priority of their aggregate scores $\gamma(e_R, e_S)$. Then, the spatial distance join is evaluated for the corresponding aR-tree nodes and the results are inserted to H_e if they are non-leaf entries (branching condition at Line 8). Otherwise, if a leaf node entry pair (r, s) (i.e., object pair) is de-heaped, it is guaranteed that (r, s) has higher aggregate score than any other object pair to be found later, since entry and object pairs are accessed in decreasing order of their γ-scores from H_e. Therefore the object pair is output as the next result of the k-SDJ query (Line 14). DFA terminates after k results have been computed.

As an example of DFA, consider the two aR-trees for the R and S datasets of Figure 2, as shown in Figure 3. Note that the aR-tree entries are augmented with the maximum scores of any objects in the subtrees indexed by them (e.g., R_2 has score 0.6). Assume that $k = 1$, $\epsilon = 0.1$, and $\gamma = SUM$. DFA begins by joining the two root nodes of A_R and A_S, which adds two entry pairs to H_e; (R_1, S_1) and (R_2, S_2); the other two combinations (R_1, S_2) and (R_2, S_1) are pruned by the ϵ-distance join predicate. The next pair to be examined is (R_1, S_1) because $\gamma(R_1, S_1) = 1.8 > \gamma(R_2, S_2)$. Thus, the nodes pointed by R_1 and S_1 are synchronously visited and their ϵ-distance join adds to H_e pairs (R_3, S_4), (R_4, S_4), and (R_4, S_3). The next entry pair to be de-heaped is (R_3, S_4) with $\gamma(R_3, S_4) = 1.7$; this results in object pair (r_1, s_6) being found and added to H_e. Next, (R_4, S_3) is de-heaped and (r_3, s_3) is added to H_e. The next pair to be popped from H_e is the object pair (r_3, s_3); note that this is guaranteed to be the top ϵ-distance join pair, since it is the first object pair to be extracted from H_e, and thus, the algorithm terminates.

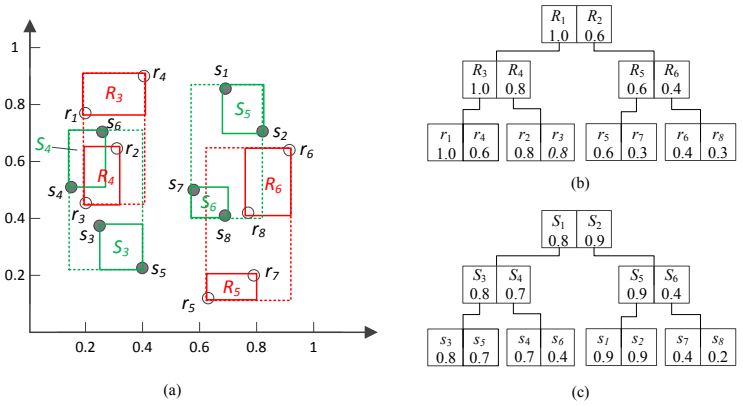


Fig. 3. Two aR-trees for R and S

DFA shares some common elements with the probabilistic spatial join algorithm of [11]. Adapting the solution proposed in [11] for k-SDJ would involve extending the classic plane-sweep approach to operate on spatially grouped objects, where group MBRs are annotated by aggregate probability values (used for pruning group pairs during evaluation). However, our approach is more optimized in the sense that it is applied on hierarchically multiple levels of grouping (instead of a single grouping level in [11]). Thus, DFA represents the best possible implementation of the framework of [11] for k-SDJ queries, which as we show in Section 4 is inferior to our best approach.

DFA is expected to be fast if the locations of objects are correlated with their atomic scores; in this case, the objects with the highest scores that are close to each other are identified fast; otherwise, many tree node pairs may have to be accessed and joined until DFA can terminate. In other words, since the data are primarily clustered by their locations in the aR-trees, pruning pairs of non-leaf node entries is mostly due to the spatial predicate while the aggregate scores may be too uniform to facilitate in this task.

3.3 The Block-based Algorithm (BA)

SFA and DFA both have certain shortcomings; on one hand, SFA fails to exploit the spatial domain to locate fast spatial join results; on the other hand, DFA does not necessarily find pairs of objects with high aggregate scores fast. Our third solution is a *Block-based Algorithm* (BA), which alleviates these shortcomings. Like SFA, BA examines the data by prioritizing objects with high scores first, however, it does not incrementally build and join the aR-trees for them (which is costly due to the repeated insertion and range query operations); instead it bulk-loads aR-trees for blocks of objects from R (or S) and spatially joins them with the blocks from S (or R) read so far, avoiding unnecessary block or node pair joins based on their score and distance bounds.

Algorithm 3. Block-based Algorithm (BA)

Input: k, ϵ, γ, R, S
Output: C
1: initialize a min-heap C:=$\emptyset$ of candidate results; initialize θ:=$-\infty$
2: sort R and S based on score, if not already sorted
3: **while** more blocks of objects exist in R and S **do**
4: i := next input to be accessed //either R or S
5: j := other input //either S or R
6: b := *get_next_block*(i) //get next block of objects from input i
7: A_b := bulk-load aR-tree for b
8: **for** each block b' of j **do**
9: **if** $\gamma(b^u, b'^u) > \theta$ **then**
10: apply DFA to join A_b with A'_b
11: retrieve results and update C and θ
12: T:=max$\{\gamma(b^u_{R1}, b^l_{Slast}), \gamma(b^l_{Rlast}, b^u_{S1})\}$
13: **if** $T \leq \theta$ **then**
14: **break** //result secured; no need to access more objects
15: **return** C

More specifically, BA (like SFA) considers R and S sorted by score. It then divides them into blocks of λ objects each. At each step a block of objects b_R (or b_S) is accessed from R (or S) and then, joined with the blocks of S (or R), which have already been accessed. Thus, BA can be considered as an adaptation of SFA that operates at the block-level. For example, Figure 4 illustrates the blocks for collections R and S if $\lambda = 2$. BA would, for instance, first read b_{R1}, then read b_{S1} and join it with b_{R1}, then read b_{S2} and join it with b_{R1}, then read b_{R2} and join it with b_{S1} and b_{S2}, etc. Since the objects in R and S are sorted in decreasing order of their scores, for each block b, there is an upper score bound b^u corresponding to the score of the first object in b. Therefore, when considering the join between two blocks b_R and b_S, $\gamma(b^u_R, b^u_S)$ represents an upper score bound for all ϵ-distance join pairs in $b_R \times b_S$. If we have found at least k distance join pairs so far, then we know that joining b_R with b_S is pointless when $\gamma(b^u_R, b^u_S) \leq \theta$, where θ is the k-th best aggregate score. In other words, the current block, e.g., b_R, is only joined with the blocks b_S of S for which $\gamma(b^u_R, b^u_S) > \theta$. For the block-level join, we employ the DFA algorithm; an aR-tree is constructed for the current block and joined with the aR-trees of blocks read from the other input using Algorithm 2.

Algorithm 3 is a high-level pseudocode for BA. Line 4 chooses the input (i.e., R or S) from which the next block will be retrieved using the same policy as SFA (i.e., the input with the highest last seen score). Without loss of generality, assume that $i = R$. Then, for the next block b accessed from R, an aR-tree is created, and b is joined with all blocks b' from S which have already been examined using DFA (for these blocks, the corresponding aR-trees have already been constructed before). The blocks b' are joined in decreasing order of their score ranges (e.g., first b_{S1}, then b_{S2}, etc.). Each new ϵ-distance join pair is used for updating C and θ (in fact the current θ is used to terminate the block-wise DFA as soon as the top pair on the heap H^e has aggregate score at most θ). After handling b, the termination threshold T is updated (Line 12). Note that this threshold is the same as in the SFA algorithm: the maximum value between $\gamma(b^u_{R1}, b^l_{Slast})$ and $\gamma(b^l_{Rlast}, b^u_{S1})$, where b^u_{R1} (b^u_{S1}) is the highest score in the first

	id	loc	$score$
b_{R1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

	id	loc	$score$
b_{S1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

Fig. 4. Example of BA

block of R (S) (i.e., the score of the very first object in the collection) and b^l_{Rlast} (b^l_{Slast}) is the lowest score in the last block of R (S) (i.e., the score of the last object in it).

As an example of BA, consider the k-SDJ ($k = 1$, $\epsilon = 0.1$, $\gamma = SUM$) between collections R and S of Figure 4, which are partitioned into blocks. Initially, b_{R1} is read and an aR-tree A_{R1} is created for it. Then, b_{S1} is accessed, bulk-loaded to an A_{S1}, and joined using A_{R1} (Line 10 of BA), producing no spatial join results. Next, b_{S2} is accessed and (unsuccessfully) joined with b_{R1}. After reading b_{R2} the block is joined with b_{S1} and b_{S2} (in this order), to generate $C = \{(r_3, s_3)\}$ and set $\theta = 1.6$. The next block b_{S3} is joined with b_{R1}, but not b_{R2}, because $\gamma(b^u_{R2}, b^u_{S3}) = \gamma(0.8, 0.7) \leq \theta$; this means that in the best case a spatial distance join between b_{R2} and b_{S3} will produce a pair with score 1.5, which is not better than the current top pair's (r_3, s_3) score. The join between b_{S3} and b_{R1} does not improve the current k-SDJ result. At this stage, BA terminates because $T = 1.5$, which is not higher than $\theta = 1.6$.

Although BA is reminiscent to SFA in that it examines the records in order of their scores, BA has two main advantages compared to SFA. First, performing the joins at the block level is more efficient, because the aR-trees for the blocks are created just once efficiently by bulk-loading (instead of iterative insertions as in SFA) and could be used for multiple block joins (e.g., b_{R1} is joined with blocks b_{S1}–b_{S3} in our example). The joins between aR-trees are much faster compared to the record-by-record probing (i.e., index nested loops) approach of SFA. Second, in BA, the currently processed block is joined only with a small number of blocks of the other input (with the help of the upper score bounds of the blocks), while in SFA the current object is probed against the entire set of objects buffered from the other input.

4 Experimental Evaluation

In this section, we present an experimental evaluation of our techniques for k-SDJ. Section 4.1 details the setup of our analysis. Sections 4.2 and 4.3 experimentally prove the superiority of SFA and DFA compared to alternative implementations that follow the score-first, the distance-first evaluation paradigm, respectively. Section 4.4 carries out a comparison between the SFA, DFA and

Table 2. Experimental parameters

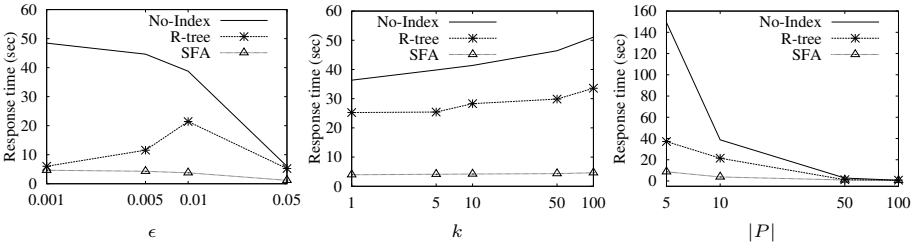
Parameter	Values	Default value
ϵ	0.001, 0.005, 0.01, 0.05	0.01
k	1, 5, 10, 50, 100	10
$\|P\|$	5, 10, 50, 100	10
$\lambda/\|R\|$	0.0005, 0.001, 0.005, 0.01, 0.02	0.005

BA algorithms. All algorithms involved in this study are implemented in C++ and the experiments were conducted on a 2.3 Ghz Intel Core i7 CPU with 8GB of RAM running OS X.

4.1 Setup

Our experimental analysis involves two collections of real spatial objects: (1) FLICKR that contains 1.68M locations associated with photographs taken from the city of London, UK over a period of 2 years and hosted on the Flickr photosharing website, and (2) ISLES that contains 20M POIs in the area of the British isles drawn from the OpenStreetMap project dump. To perform the experiments, every collection is split into two equally sized parts denoted by R and S. This is to avoid performing a self-join which will produce result pairs involving exactly the same object. Since real scores for objects were not available, we generated them as follows. For each part, we first randomly created $|P|$ seed points (simulating POIs) and generate the object scores with the following formula: $o.score = 1 - \min_{i=1}^{|P|}[dist(o, P_i)]$. Intuitively, real life objects are more likely to have high scores if they are close to a POI [18]. For example, the hotels close to the city center have higher potential to be highly rated for their convenience. The generated scores are normalized to take values in $[0, 1]$. In our study, we vary the total number of seed points created for each of R and S, from 5 to 100. Larger values of $|P|$ tend to generate more uniform score distributions and higher average scores. Finally, note that we consider SUM as the score aggregation function γ for our experiments.

To assess the performance of each join method, we measure their response time that also includes index building cost and all sorting costs, varying (1) the distance threshold ϵ, (2) the number of returning pairs k, and (3) the number of seed points $|P|$. Further, in case of BA we also vary the ratio $\lambda/|R|$ of the block size λ over the size of the collection $|R|$ ($|S|$). Table 2 summarizes all the parameters involved in this study. On each experiment, we vary one of ϵ, k, $|P|$ and $\lambda/|R|$ while we keep the remaining parameters fixed to their default values. Finally, note that both the collections and the indexing structures used by the evaluation methods are stored in main memory (like relational top-k join studies, we consider k-SDJ evaluation in main memory), and that we set the page size for both the R-trees and the aR-trees to 4KB.



(a) Distance threshold (b) Number of results (c) Number of seed points

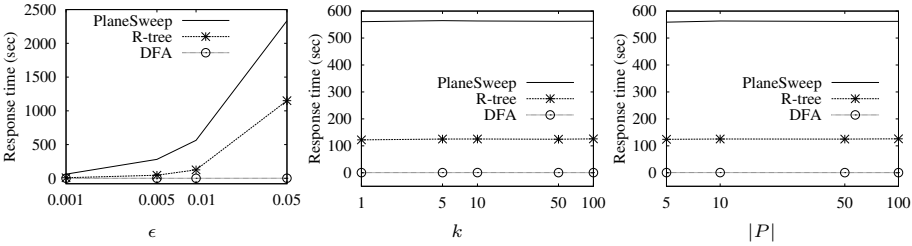
Fig. 5. Comparison of the score-first algorithms on FLICKR

4.2 Score-First Algorithms

The first set of experiments demonstrates that our SFA algorithm, as described in Section 3.1, is an efficient adaptation of the state-of-the-art top-k relational join algorithm HRJN. Recall that, in SFA, we use aR-trees to index the objects seen so far from R and S and use an efficient aR-tree search algorithm, which exploits the aggregate scores stored at intermediate entries to prune the search space while searching for join pairs for the current object o. In fact, the score-first paradigm could also be applied without using aR-trees for indexing (in this case, a scan would be applied against the buffered objects to find the spatial join pairs), or one could use R-trees instead of aR-trees. Figure 5 compares the response times of these two alternatives with SFA (denoted by No-Index and R-tree, respectively), while varying ϵ, k and $|P|$. As expected, SFA outperforms the other two methods since it is able to prune object pairs in terms of both their spatial distance and their aggregate scores. We also observe that increasing ϵ makes the k-SDJ evaluation faster for SFA and No-Index, but not for R-tree. The reason is the following. As ϵ increases, more object pairs qualify the spatial predicate. Since the objects are sorted in descending order of their scores, a smaller number of pairs needs to be examined. Although this holds also for R-tree, its time initially increases due to the increasing cost of the range queries involved. Another important observation is that the response time of SFA is less affected by the increase of k compared to the other methods. Particularly, for $k = 100$, SFA needs 10% more time to compute k-SDJ compared to $k = 1$, while in case R-trees or No-Index, this increase is 33% and 45%, respectively. Finally, with the usage of more seed nodes for score generation, the response time of all methods decreases since more object pairs have high aggregate scores.

4.3 Distance-First Algorithms

We perform a similar evaluation for DFA, by comparing it against two algorithms that adopt alternative techniques for a distance-first k-SDJ processing (instead of building and joining two aR-trees). Thus, one option is to generate and join two R-trees (on each of R and S), and another option is to sort R and S an apply



(a) Distance threshold (b) Number of results (c) Number of seed points

Fig. 6. Comparison of the distance-first algorithms on FLICKR

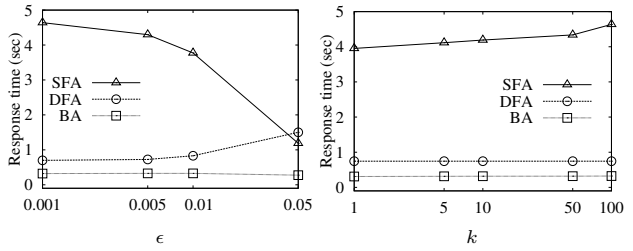
the plane sweep spatial join technique. Figure 6 compares the three approaches, showing that their response time is affected only by the increase of the distance ϵ and not by k or $|P|$; this is due to the fact that all methods primarily focus on the spatial predicate of the k-SDJ, which is independent of the scores. Due to its ability to use score aggregation bounds, DFA not only outperforms the other methods (in some cases for more than two orders of magnitude), but it is also very little affected by the increase of ϵ.

We also implemented BA with different versions of its block-based join module; i.e., the aR-tree based join as described in Section 3.3, and alternatives based on plane sweep and R-tree join. The results (not included due to space constraints) confirm that the aR-tree based module for joining blocks is superior to the other alternatives (the trends are similar to the experiments of Figure 6).

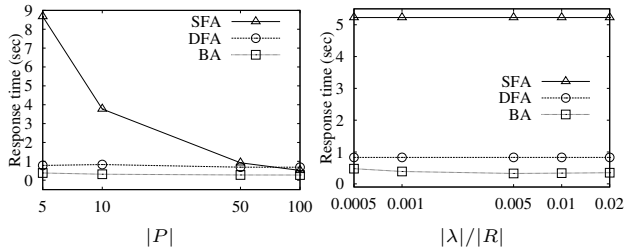
4.4 Comparison of the Evaluation Paradigms

We have already shown that SFA, DFA, and BA, as described in Section 3 are efficient implementations for the corresponding search paradigms (search-first, distance-first, and block-based, respectively). We now compare these three algorithms to identify the best evaluation paradigm and algorithm for k-SDJ. Figure 7 and 8 report on their response time for FLICKR and ISLES[3], respectively, while varying ϵ, k, $|P|$ and $\lambda/|R|$. The figures show that BA outperforms SFA and DFA in all cases. It is important to also notice that BA is very robust to the variation of the parameters. Specifically, its response time is always around 350msec for FLICKR and 3.5sec for ISLES. In contrast, SFA is (1) positively affected by the increase of ϵ since more object pairs qualify the spatial predicate and thus, the top-k results can be quickly identified, and (2) negatively by the increase of k as it needs to examine more object pairs. Further, DFA becomes slower with ϵ as it primarily focuses on the spatial predicate of the k-SDJ. BA manages to combine the above advantages of the score-first paradigm and SFA, and the distance-first paradigm and DFA, as it examines blocks of objects ordered by score and applies the spatial predicate at the block level instead on the whole collections.

[3] In this experiment we used only half of the ISLES collection; i.e., $|R| = |S| = 5$M.

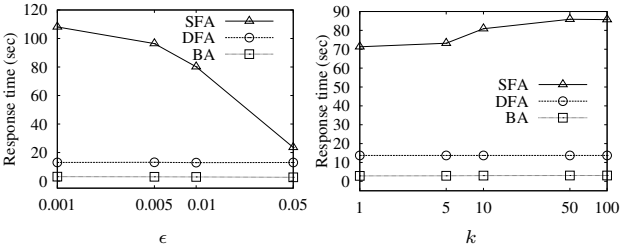


(a) Distance threshold (b) Number of results

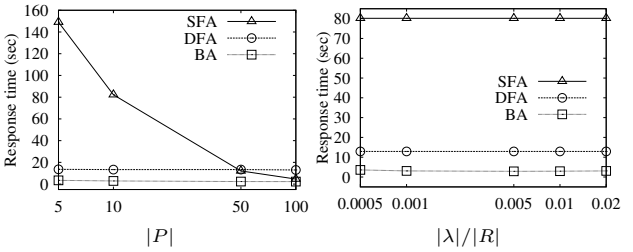


(c) Number of seed points (d) Block size ratio

Fig. 7. Comparison of SFA, DFA and BA algorithms on FLICKR



(a) Distance threshold (b) Number of results



(c) Number of seed points (d) Block size ratio

Fig. 8. Comparison of SFA, DFA and BA algorithms on ISLES

We also performed a scalability experiment, by joining samples R and S of the ISLES dataset of different sizes $|R|$ and $|S|$, while setting ϵ, k, $|P|$ and $\lambda/|R|$ to their default values. Figure 9(a) shows the results of the scalability test. We observe that all methods scale similarly, with BA being 1-2 orders of magnitude faster than the other methods. Note that even on the full ISLES collection, BA needs less than 7 seconds to evaluate a k-SDJ query.

In the last experiment, we evaluate the performance of the methods on datasets of different cardinalities. R and S are samples of the ISLES dataset of varying cardinality ratio $|R| : |S|$, while $|R| + |S|$ is fixed to 10M. The result is shown in Figure 9(b). We observe that the response times of all methods only grow slightly with the increase of $|R| : |S|$ (the effect is more obvious on SFA). This is because although the cardinality ratio is changing, the percentage of traversed records on both size does not change much. Still, maintaining and updating a larger aR-tree on R costs a little more than what is saved on a smaller aR-tree for S, leading to the slight cost growth as the ratio increases.

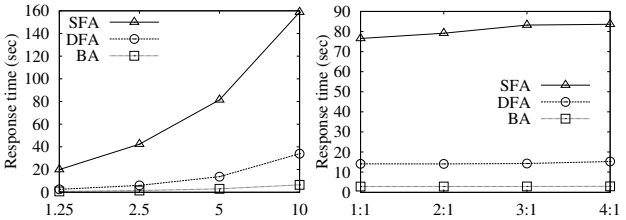


Fig. 9. Scalability and Cardinality test on ISLES

5 Conclusion

In this paper, we proposed and studied top-k spatial distance join (k-SDJ) queries. Although this operator finds practical application, it has not been studied in the past, therefore no efficient solutions exist for it, so far. We proposed three algorithms for k-SDJ queries. SFA accesses the objects from the joined collections incrementally and spatially joins them to find new results, using bounds to terminate early. DFA incrementally computes the distance join results ordered by score, by extending a spatial join algorithm to apply on aggregate R-trees that index the two inputs. BA is a hybrid approach, which considers blocks of objects from the two inputs, ordered by score, and joins them as necessary, until a termination condition is reached. Our experimental findings on large datasets show that BA performs best in practice, while the optimized use of aR-trees in all methods greatly improves their performance compared to baseline alternatives which rely on simpler indexing schemes. A direction for future work is to extend our algorithms to compute top-k object pairs according to a combined distance-based and score-based aggregate function.

References

1. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Scalable sweeping-based spatial join. In: VLDB, pp. 570–581 (1998)
2. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using R-trees. In: SIGMOD Conference (1993)
3. Chan, E.P.F.: Buffer queries. IEEE Trans. Knowl. Data Eng. 15(4), 895–910 (2003)
4. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. In: SIGMOD Conference (2000)
5. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS, pp. 102–113 (2001)
6. Hjaltason, G.R., Samet, H.: Incremental distance join algorithms for spatial databases. In: SIGMOD Conference (1998)
7. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. 24(2), 265–318 (1999)
8. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. In: VLDB, pp. 754–765 (2003)
9. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40(4) (2008)
10. Leutenegger, S.T., Edgington, J.M., Lopez, M.A.: STR: A simple and efficient algorithm for R-tree packing. In: ICDE, pp. 497–506 (1997)
11. Ljosa, V., Singh, A.K.: Top-k spatial joins of probabilistic objects. In: ICDE, pp. 566–575 (2008)
12. Mamoulis, N., Yiu, M.L., Cheng, K.H., Cheung, D.W.: Efficient top-k aggregation of ranked inputs. ACM Trans. Database Syst. 32(3) (2007)
13. Natsev, A., Chang, Y.C., Smith, J.R., Li, C.S., Vitter, J.S.: Supporting incremental join queries on ranked inputs. In: VLDB, pp. 281–290 (2001)
14. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP operations in spatial data warehouses. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 443–459. Springer, Heidelberg (2001)
15. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD Conference (1995)
16. Schnaitter, K., Polyzotis, N.: Optimal algorithms for evaluating rank joins in database systems. ACM Trans. Database Syst. 35(1) (2010)
17. Shin, H., Moon, B., Lee, S.: Adaptive multi-stage distance join processing. In: SIGMOD Conference (2000)
18. Yiu, M.L., Lu, H., Mamoulis, N., Vaitis, M.: Ranking spatial data by quality preferences. IEEE Trans. Knowl. Data Eng. 23(3), 433–446 (2011)
19. Zhu, M., Papadias, D., Lee, D.L., Zhang, J.: Top-k spatial joins. IEEE Trans. Knowl. Data Eng. 17(4), 567–579 (2005)