Noname manuscript No.
(will be inserted by the editor)

Top-k Spatial Distance Joins

Shuyao Qi · Panagiotis Bouros · Nikos Mamoulis

Received: date / Accepted: date

Abstract Top-k joins have been extensively studied when numerical valued attributes are joined on an equality predicate. Other types of join attributes and predicates have received little to no attention. In this paper, we consider spatial objects that are assigned a score (e.g., a ranking). Give two collections R, S of such objects and a spatial distance threshold ϵ, we introduce the top-k spatial distance join (k-SDJoin) to identify the k pairs of objects, which have the highest combined score (based on an aggregate function γ) among all object pairs in $R \times S$ with a spatial distance at most ϵ. State-the-of-art methods for relational top-k joins can be adapted for k-SDJoin, but their focus is on minimizing the number of objects accessed from the inputs; however, when spatial objects are joined, the computational cost can easily become the bottleneck. In view of this, we propose a novel evaluation algorithm, which greatly reduces the computational cost, without compromising the access cost. The main idea is to access and efficiently join blocks of objects from each collection, using appropriate bounds to avoid computing the entire spatial ϵ-distance join. As the performance of our solution heavily relies on the size of the input blocks, we devise an approach for automated block size tuning enhanced by a novel generic model for estimating the number of objects to be accessed from each input. Contrary to previous efforts, our model employs cheap-to-

S. Qi
Department of Computer Science
The University of Hong Kong
E-mail: qisy@connect.hku.hk

P. Bouros
Institute of Computer Science
Johannes Gutenberg University Mainz, Germany
E-mail: bouros@uni-mainz.de

N. Mamoulis
Department of Computer Science & Engineering
University of Ioannina, Greece
E-mail: nikos@cs.uoi.gr

compute statistics and requires no prior knowledge of data distribution. Our extensive experimental analysis demonstrates the efficiency of our algorithm compared to methods based on existing literature that prioritize either the ranking or the spatial join component of k-SDJoin queries.

Keywords Top-k join · spatial join

1 Introduction

Ranking has been widely considered by the data management community in the context of top-k queries [8,14,23,20] and top-k joins [7,25,27,36,37,41, 45]. In the latter, ranked inputs are joined to derive objects or tuple pairs which maximize an aggregate function on scoring attributes. For instance, the following SQL query joins relations Customers and Suppliers to return the top-10 (i.e., $k = 10$) retail sales of a company that maximize the associated profit defined as the amount paid by a customer plus the discount offered by a supplier:

```
SELECT *
FROM Customers C, Suppliers S
WHERE C.productID = S.productID
ORDER BY (C.price + S.discount) DESC
LIMIT 10;
```

Essentially, the evaluation of top-k joins has been studied only for relational data objects with join attributes of primitive data types such as numerical values, and for an equality join predicate. Other data types and join predicates have received no attention.

With the rapid advances in location sensing technologies, spatial data are nowadays ubiquitous. The spatial join operator retrieves pairs of objects that satisfy a spatial predicate. Although spatial joins have been extensively studied during the last two decades [1,3,16,22,26,29,34,44], due to their applicability and high execution cost, this query type focuses solely on spatial attributes, while in many applications spatial objects have additional attributes. For instance, restaurants shown in websites like TripAdvisor and Yelp are assigned user-generated ratings or spatial objects on emerging scientific fields like atmospheric, oceanographic, and environmental sciences are associated with measurements varying from temperature and pressure to earth's gravity and seismic activity. In other words, despite the vast availability of spatial objects associated with such numerical (for simplicity, say scoring) attributes, to our knowledge, there exists few join operator that considers both location and scoring attributes at the same time.[1]

On an attempt to fill this gap, we introduce the *top-k spatial distance join* (k-SDJoin). Given two collections of spatial objects R and S that also carry

[1] An exception is the work of [21] which, however, is restricted to a specific type of attributes (probabilities) and a specific aggregation function (product).

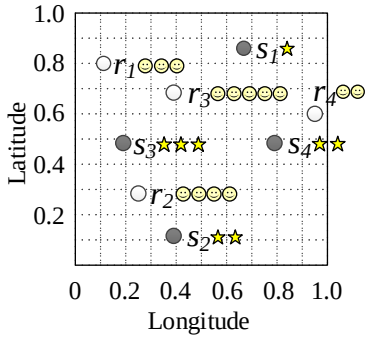


Fig. 1 Example of a top-k spatial distance join

a scoring attribute `score`, a k-SDJoin 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 location attributes `loc` of r and s), and for every $(r', s') \in R \times S \setminus J$ with $dist(r', s') \leq \epsilon$, $\gamma(r, s) \geq \gamma(r', s')$ holds, where γ is a *monotone aggregate function* (e.g., SUM) which combines the scoring attributes `score` of two objects. As an exemplary application, k-SDJoin could recommend to the visitors of a city the k pairs of closely located restaurants and hotels with the highest combined ratings. Figure 1 shows collection R of four restaurants $\{r_1, \ldots, r_4\}$ and S of four hotels $\{s_1, \ldots, s_4\}$; every object carries a score shown next to its location. Assuming that qualifying pairs should have an Euclidean distance at most $\epsilon = 0.3$ and $\gamma = SUM$, 2-SDJoin returns (r_2, s_3) with aggregate score 7 and (r_2, s_2) with score 6. Notice that although $dist(r_4, s_4) < \epsilon$, pair (r_4, s_4) is not included in the query result as $\gamma(r_4, s_4) = 4 < \gamma(r_2, s_2) < \gamma(r_2, s_3)$. 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 distance to r_3 smaller than 0.3. k-SDJoin also finds application in emerging scientific fields such as bioinformatics; e.g., for investigating protein stability. For instance, bioinformaticians could employ k-SDJoin to identify amino acid pairs that are close to each other with respect to their 3D location, while having good properties which can be quantified as scores, e.g., the solvent accessibility [30].

To evaluate k-SDJoin, we first consider existing literature for rank and spatial joins. On one hand, state-of-the-art methods for top-k joins assume that objects from the input collections R and S can be accessed in decreasing order of their `score` attribute; hence, k-SDJoin can be evaluated by accessing R and S only partially, using bounds for the non-examined objects to terminate [14,25,37]. In this spirit, Section 2.2 presents the *Score-First Algorithm* (SFA); each time $r \in R$ (or $s \in S$) is accessed by SFA, the object is joined with previously accessed objects from S (or R). The accessed objects are buffered and indexed on their spatial location attribute `loc`. On the other hand, if data are centralized (the case we focus on this paper), an alternative evaluation approach for k-SDJoin is to primarily focus on the join component of the query,

i.e., compute the $R \bowtie_\epsilon S$ spatial ϵ-distance join entirely, and then identify the k object pairs with the highest aggregate score. This *Distance-First Algorithm* (DFA) is presented in Section 2.3.

Previous work for top-k joins primarily focused on minimizing the number of objects accessed from the input collections, which corresponds to I/O cost in a centralized setting (assuming unlimited memory for buffering/indexing the accessed objects) [14,20,23,25,37] or to network cost when inputs (or parts thereof) reside on different machines [7,41,45]. For the relational top-k equijoins studied by these works, the computational cost is very low because buffered objects are indexed by hash tables, which support search and updates in constant time. However, in the case of k-SDJoin, our analysis in Section 6.2 shows that the computational cost of SFA is up to several times higher than the access cost, due to the increased overhead of searching and incrementally updating spatial indexing structures, e.g. R-trees. In view of this, we propose the *Block-based Algorithm* (BA) for the efficient evaluation of k-SDJoin, in terms of the CPU cost.

BA processes spatial objects in decreasing order of their scoring attribute similar to SFA but in a block-wise fashion, and joins blocks of the input collections similar to DFA but uses score bounds to avoid computing the entire $R \bowtie_\epsilon S$ join. Despite focusing on binary top-k join under one scoring attribute per input, BA can incorporate the pulling strategy and the bound scheme from [10] for multiple scores (discussed in Section 3.4), and handle multiple inputs either in a multi-way fashion [37] or as a hierarchy of binary k-SDJoin operators [20] (discussed in Section 5).

The performance of BA is strongly related to the size λ of the blocks accessed from the inputs; i.e., the number of contained objects. Under this, we introduce the *objective cost function* $\mathcal{C}(\lambda)$ to capture the total cost of computing k-SDJoin with BA, and model the selection of the most appropriate block size as an optimization problem. We also devise a novel model for estimating the number of objects accessed from each input collection which, in contrast to previous models of [7,15,20,38], employs cheap-to-compute statistics and requires no prior knowledge of the join/scoring attribute distribution. In fact, due to employing expensive statistics, i.e., multi-dimensional histograms, the models of [7,38] can be employed only for relational top-k joins.

Contributions. In brief, the key contributions of our work are summarized as follows:

- We introduce k-SDJoin for spatial objects with scoring attributes. The k-SDJoin query can be used either as a standalone operation or participate in complex query evaluation plans. For this purpose, we assume that the input collections are not indexed in advance.
- We present SFA and DFA for evaluating k-SDJoin based on existing literature; the algorithms prioritize either the ranking or the spatial joining component of the query, respectively.
- We propose BA, which performs block-wise evaluation, combining the benefits of SFA and DFA, without sharing their shortcomings.

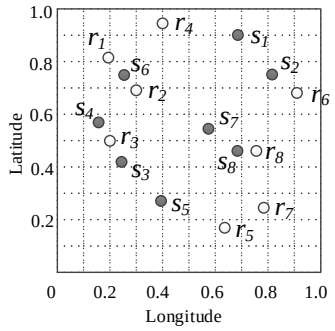
- We optimize all three algorithms by employing aR-trees [28] as spatial indexing (albeit in different fashions) in order to combine spatial search with score-based pruning.
- We elaborate on the instance optimality of BA building upon the analysis in [37] for relational top-k joins.
- We devise an approach for automated block size tuning, which effectively guarantees the performance of BA.
- We devise a novel generic model for estimating access depth from the input collections; besides the block size selection for BA, our model can be applied to all depth estimation tasks of rank join queries.
- We conduct an extensive experiment analysis using both real-world and synthetic collections to investigate the efficiency of the evaluation methods, and the effectiveness and the accuracy of our proposed models.

Comparison to previous publication. This article is an extension of our preliminary work presented in [32]. The additional technical contributions include the performance analysis of BA, the automated block size tuning approach for BA, and the model for access depth estimation. Besides, we include a more detailed and comprehensive description of the algorithms and their pruning techniques, and prove their correctness. Finally, we extend our experimental study (i) using real-world datasets, (ii) including new tests to investigate the effectiveness of automatic block size tuning and the accuracy of our depth estimation model, and (iii) repeating all tests on BA as its block size is now automatically set. The results are consistent with our preliminary tests in [32].

Outline. The remainder of this article is organized as follows. Section 2 presents the SFA and DFA algorithms for k-SDJoin evaluation based on existing literature, while Section 3 presents our novel algorithm BA. Next, Section 4 details our models for automatic block size tuning on BA and estimating the number of accessed objects, while Section 5 elaborates on the multiple inputs setup of k-SDJoin. Our experimental analysis is reported in Section 6. Last, Section 7 reviews related work, and Section 8 concludes the article and discusses future work.

2 Evaluation Based on Existing Literature

According to the definition in Section 1, k-SDJoin identifies pairs of objects (i) positioned in nearby locations (ii) with a high aggregate score based on a monotone aggregate function γ. Hence, k-SDJoin comes as a combination of a spatial distance join and a top-k query. We next present two evaluation methods inspired by existing literature that prioritize either of the two k-SDJoin sub-queries/components. Section 2.2 primarily considers the scoring attribute and ranking while Section 2.3 prioritizes the distance join predicate. Table 1 summarizes the notation used throughout the rest of this article, while Figure 2 illustrates our running example with the collections of spatial objects



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)	0.6
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 Running example of collections R and S with 8 objects each.

Table 1 Notation used throughout the article.

notation	description
R/S	Input object collection
k	Number of required results
ϵ	The spatial distance join threshold
$dist(r, s)$	Euclidean distance between the spatial location of objects r and s
γ	A monotone aggregate function
$\mathcal{A}_R/\mathcal{A}_S$	aR-tree on an input object collection
C	Candidate/result set of k-SDJoin
θ	k-th highest aggregate score; i.e., lowest aggregate score in C
r_i/s_j	The i-th/j-th object in a sorted collection
c_R/c_S	Any-k depth; i.e., number of accessed objects to find the first k candidates
d_R/d_S	Top-k depth; i.e., number of accessed objects to compute k-SDJoin
b_R/b_S	A block of objects from an input collection
λ	Size of a block; i.e., number of contained objects
b^u_R/b^u_S	Upper score bound of a block; i.e., highest object score in the block
b^ℓ_R/b^ℓ_S	Lower score bound of a block; i.e., lowest object score in the block

$R = \{r_1, ..., r_8\}$ and $S = \{s_1, ..., s_8\}$. For simplicity, we consider the Euclidean distance between two objects as $dist(r, s)$ but our analysis can be extended to other distance measures, e.g., shortest distance in spatial road networks.

2.1 Enhanced Spatial Indexing

The dominant indexing structure for spatial data is the R-tree [11]; however both methods presented next perform better when employing (in a different fashion each) an extension to the R-tree, called the aR-tree [28]. The aR-tree has identical structure and update algorithms to 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 aR-trees built for the collections of Figure 4. Note that the entries are augmented with the maximum scores of all objects in the subtrees indexed by them (e.g., R_2 has score 0.6). The aR-tree prioritizes k-SDJoin according to the aggregate score and allows for pruning on both the spatial distance join predicate and the aggregate score. Our tests

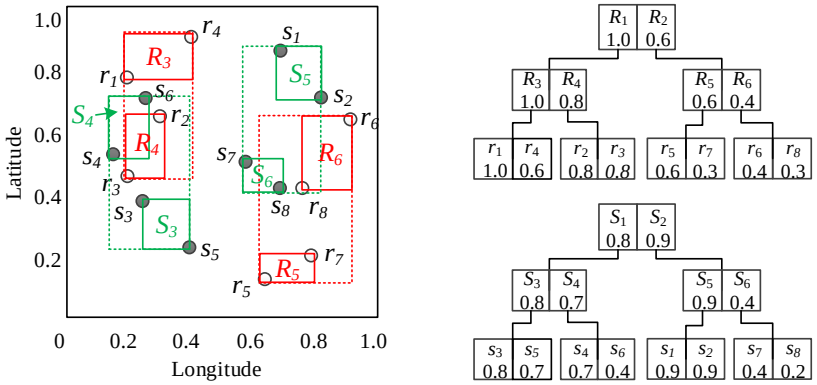


Fig. 3 aR-trees for collections R and S of Figure 2.

in Section 6.2 demonstrate the advantage of the aR-tree based evaluation over the R-tree based and plane-sweep.

2.2 The Score-First Algorithm

Ilyas et al. [14] proposed the *Hash-based Rank-JoiN* (HRJN*) binary operator for top-k joins, which progressively accesses input collections R and S in order of their `score` attributes, and incrementally produces results. HRJN* joins the currently accessed object, e.g., from R, on join attribute with the buffered (i.e., previous accessed) objects from S, which are indexed by a hash-table. Join results are organized in a priority queue based on their aggregate score. Let ℓ_R, h_R (ℓ_S, h_S) be the lowest and highest scores seen in collection R (S) so far; all join results currently in the queue with aggregate score higher than *threshold* $T = \max\{\gamma(h_R, \ell_S), \gamma(\ell_R, h_S)\}$ are guaranteed to have higher aggregate score than every future join result and thus can incrementally be output as top-k join results. To study the optimality of rank-join algorithms in more general problems of multiple inputs with one or more scoring attributes, Schnaitter and Polyzotis [37] introduced the Pull/Bound Rank Join (PBRJ) evaluation framework; basically, HRJN* is an instantiation of PBRJ, denoted by PBRJ$^*_c$, which applies the threshold-adaptive pulling strategy for accessing objects and the so-called corner bound scheme for computing the threshold T.

We next discuss the *Score-First Algorithm* (SFA), which prioritizes the ranking sub-query/component of k-SDJoin. Essentially, the algorithm adapts binary HRJN*/PBRJ$^*_c$ to work with a spatial ϵ-distance join predicate. Algorithm 1 is a high-level pseudo-code of SFA, which takes as input two object collections R and S, spatial distance join threshold ϵ, a monotone aggregate function γ on their scoring attributes `score`, and an integer k, i.e., the number of requested results. First, in Lines 1–2, the algorithm sorts (if needed) [2] inputs

[2] Input collections R and S need not to be sorted on their scoring attribute for example, if they stem from previous query operators which produce such interesting orders.

Algorithm 1: Score-First Algorithm (SFA)

Input : object collections R, S, spatial distance join threshold ϵ, monotone aggregate function γ, number of results k
Output : result set $C = \{(r, s) \in R \times S : dist(r, s) \leq \epsilon\}$
Variables : aR-trees $\mathcal{A}_R$, $\mathcal{A}_S$, bound θ, termination threshold T, the lowest seen scores ℓ_R, ℓ_S

1 **initialize** $C \leftarrow \emptyset$, $\theta \leftarrow -\infty$, $\mathcal{A}_R \leftarrow \emptyset$, $\mathcal{A}_S \leftarrow \emptyset$, $\ell_R \leftarrow \infty$, $\ell_S \leftarrow \infty$;
2 **sort** R and S in descending order of attribute `score`; ▷ `If not already sorted`
3 **while** more objects exist in R and S **do**
4 $i \leftarrow S$, **if** $\ell_S > \ell_R$; **otherwise** R ; ▷ `Determine current input`
5 $j \leftarrow R$, **if** $\ell_S > \ell_R$; **otherwise** S;
6 $o_i \leftarrow$ GetNextObject(i); ▷ `Get next object from current input`
7 $\ell_i \leftarrow o_i$.`score`; ▷ `Update the lowest seen score from current input`
8 $T \leftarrow \max\{\gamma(h_R, \ell_S), \gamma(\ell_R, h_S)\}$; ▷ `Update HRJN* termination threshold`
9 $\langle \theta, C \rangle \leftarrow$ ProbeAndUpdateResult($o_i, \mathcal{A}_j, T, \epsilon, \gamma, k, \theta, C$);
10 **if** $T \leq \theta$ **then** ▷ `Result secured`
11 **break;**
12 **insert** o_i to $\mathcal{A}_i$; ▷ `Update aR-tree`
13 **return** C;

R, S and initializes aR-trees $\mathcal{A}_R$, $\mathcal{A}_S$, min-heap C, bound θ and the lowest seen scores ℓ_R, ℓ_S. Next, in Lines 3–12, SFA incrementally accesses objects from collection R or S and evaluates the k-SDJoin query. At each iteration, SFA first decides which collection should be accessed and consequently, which object will be examined. Following the pulling strategy of HRJN*, SFA reads the next object from the collection with the higher lowest seen score[3], i.e., the higher between ℓ_R and ℓ_S. Without loss of generality, assume that the next object to be examined is r from R (i.e., $i = R$, $j = S$ and $o_i = r$ in Lines 4, 5 and 6, respectively); the other case is symmetric. With current object r, SFA performs the following steps:

(i) It updates termination threshold $T = \max\{\gamma(h_R, \ell_S), \gamma(\ell_R, h_S)\}$ using the corner bounding scheme of HRJN* in Line 8; h_R and h_S are the highest seen scores from R and S, respectively, i.e., they equal the score of the very first object in each collection. Note that the examination order of the objects employed by SFA allows threshold T to decrease faster and hence, SFA to terminate earlier.

(ii) It probes object r against aR-tree $\mathcal{A}_S$ to retrieve objects $s \in S$, such that pair (r, s) qualifies the spatial distance join predicate $dist(r, s) \leq \epsilon$, and $\gamma(r, s) > \theta$ holds. To this end, SFA invokes the ProbeAndUpdateResult procedure in Line 9. ProbeAndUpdateResult employs $\mathcal{A}_S$ to identify every qualifying (r, s) pair and then updates C, θ as follows. If $|C| < k$, pair (r, s) is inserted into candidates set C regardless of its aggregate score. Otherwise, (r, s) is inserted into C only if $\gamma(r, s) > \theta$ and in this case, it *replaces* the k-th pair in C, such that set C always keeps the best k pairs found so far. Finally, θ is updated to the k-th aggregate score in C.

[3] When a dataset is sorted in descending order of its scoring attribute, the lowest seen score is equivalent to the last seen score.

(iii) It checks if the evaluation of the k-SDJoin query can terminate in Line 10. Specifically, as soon as $T \leq \theta$, SFA terminates reporting C as the final result.
(iv) It updates aR-tree $\mathcal{A}_R$ on collection R inserting object r (Line 12) to be probed by objects of S in future iterations.

We now elaborate on Step (ii). Procedure 1 is a pseudo-code for the ProbeAndUpdateResult procedure. Given an object o (either $r \in R$ or $s \in S$), ProbeAndUpdateResult performs a search on the aR-tree $\mathcal{A}$ of the other collection (resp. $\mathcal{A}_S$ or $\mathcal{A}_R$) as a *score-based incremental ϵ-distance range query*. The aR-tree search is guided by a max-heap H of (o, e) pairs where e is an entry of the aR-tree $\mathcal{A}$. Max-heap H allows ProbeAndUpdateResult to examine (o, e) pairs in decreasing order of their aggregate score $\gamma(o, e)$. Note that $\gamma(o, e)$ is computed using the aggregate score for entry e in the aR-tree $\mathcal{A}$ and it is an upper bound of the score of every object contained in the subtree pointed by e. During the aR-tree search, an (o, e) pair is pruned if (i) the MBR of entry e is farther than ϵ, i.e., $dist(o, e) > \epsilon$ (see footnote[4]), or (ii) $\gamma(o, e) \leq \theta$ as it would not be possible to find an object o' in the subtree pointed by e with $\gamma(o, o') > \theta$. Otherwise, pair (o, e) is inserted to the max-heap H. Finally, notice that when ProbeAndUpdateResult examines an (o, e) pair where e is a leaf node entry of the aR-tree $\mathcal{A}$, i.e., e is an object (Lines 13–15), the procedure updates the set of candidates pairs C and the θ bound which is the aggregate score of k-th candidate pair found so far.

Example 1 Consider the collections of Figure 2, the aR-trees of Figure 3 and a k-SDJoin query with $k = 1$, $\epsilon = 0.1$, and $\gamma = SUM$. First, SFA accesses r_1 from R and as the aR-tree $\mathcal{A}_S$ is currently empty, r_1 is just inserted to $\mathcal{A}_R$. Then, s_1 is accessed from S and probed against $\mathcal{A}_R$, resulting in no match as $dist(r_1, s_1) > \epsilon$. Since $\ell_R = 1.0 > \ell_S = 0.9$, object r_2 is next accessed and joined (unsuccessfully) with $\mathcal{A}_S$. Similarly, s_2 and s_3 are processed, still without producing any distance join results. When r_3 is accessed and joined with $\mathcal{A}_S$, which now contains $\{s_1, s_2, s_3\}$, SFA inserts to C the first result (r_3, s_3). As bound $\theta = \gamma(r_3, s_3) = 1.6$ and termination threshold $T = \max\{\gamma(1.0, 0.8), \gamma(0.8, 0.9)\} = 1.8 > \theta$, a possibly better pair can be found and SFA cannot terminate yet. Next accessed object is r_4, which gives no join results. When s_4 is accessed and probed against $\mathcal{A}_R$ containing $\{r_1, r_2, r_3, r_4\}$, pair (r_3, s_4) qualifies the spatial distance predicate, i.e., $dist(r_3, s_4) < \epsilon = 0.1$, but it is discarded as $\gamma(r_3, s_4) = 1.5 < \theta$. Then, s_5 gives no new join pairs. Finally, SFA retrieves object s_6 but fails to produce any join pair with an aggregate score higher than θ. At this stage, termination threshold $T = 1.5 < \theta$ and SFA terminates returning $C = \{(r_3, s_3)\}$. ■

Correctness analysis. Ilyas et al. proved in [14] that HRJN* correctly reports the top-k join results when hash join is used for relational join attributes. To

[4] In this paper, we define the *dist* function on non-leaf entries as the minimum distance between the MBR of two tree entries bounding boxes or between the MBR of a tree entry and an object, i.e., $dist(e, e') = MINDIST(e, e')$ or $dist(o, e) = MINDIST(o, e)$, respectively.

Procedure 1: ProbeAndUpdateResult

```
   Input     : object o, aR-tree A, termination threshold T, spatial distance join
               threshold ε, monotone aggregate function γ, number of results k,
               k-th highest aggregate score θ, candidate set C
   Output    : updated bound θ, candidate set C
   Variables : max-heap H of aR-tree entries, organized by aggregate scores
 1 foreach entry e in A.root do                                  ▷ Initialize heap H
 2     if dist(o, e) ≤ ε then
 3         H.push(e);
 4 while H ≠ ∅ and T > θ do
 5     e ← H.pop();
 6     if γ(o, e) ≤ θ then
 7         break
 8     if e is non-leaf node entry then
 9         n ← node of A pointed by e;
10         foreach entry e' ∈ n do
11             if γ(o, e') > θ and dist(o, e') ≤ ε then
12                 H.push(e');
13     else
14         insert (o, e) to C, remove the k-th pair in C first if |C| = k ;
15         θ ← aggregate score of the k-th pair in C;
16 return ⟨θ, C⟩;
```

prove the same for SFA and k-SDJoin, it suffices to show the correctness of Procedure 1 and in specific, the correctness of its two pruning criteria in Lines 2 and 6.

Lemma 1 *Given an object o and an aR-tree entry e, if $dist(o, e) > \epsilon$, all objects in e can be safely pruned.*

Proof $dist(o, e)$ is defined as the minimum distance between object o and entry e, i.e., $dist(o, e) = MINDIST(o, e)$. Hence, for every object $o' \in e$, we have $dist(o, o') \geq dist(o, e) > \epsilon$; therefore, e cannot contain any valid candidate to join with o. □

Lemma 2 *Given an object o and an aR-tree entry e, if $\gamma(o, e) \leq \theta$, all objects in e can be safely pruned.*

Proof Based on the definition of the aR-tree, for every object $o' \in e$, $o.score \leq e.score$. Since γ is a monotone function, we have $\gamma(o, o') \leq \gamma(o, e) \leq \theta$. Therefore, e cannot contain an object which paired to o will provide an aggregate score higher than current k-th score θ. □

Last, We introduce the following theorem for the correctness of SFA:

Theorem 1 *SFA correctly computes k-SDJoin.*

Proof The theorem follows naturally from Lemmas 1 and 2, and Theorem 4.2.1 in [14]. □

Algorithm 2: Distance-First Algorithm (DFA)

Input : object collections R, S, spatial distance join threshold ϵ, monotone aggregate function γ, number of results k
Output : result set $C = \{(r, s) \in R \times S : dist(r, s) \leq \epsilon\}$
Variables : aR-trees $\mathcal{A}_R$, $\mathcal{A}_S$

1 **initialize** $C \leftarrow \emptyset$;
2 $\mathcal{A}_R \leftarrow$ CreateIndex(R); ▷ `Bulk-load aR-tree for` R
3 $\mathcal{A}_S \leftarrow$ CreateIndex(S); ▷ `Bulk-load aR-tree for` S
4 $C \leftarrow$ Join($\mathcal{A}_R, \mathcal{A}_S, \epsilon, \gamma, k$); ▷ `Spatial` ϵ`-distance join`
5 **return** C;

2.3 The Distance-First Algorithm

The *Distance-First Algorithm* (DFA) prioritizes the spatial distance join subquery/component of a k-SDJoin query. A straightforward approach for DFA would first produce all object pairs that qualify the join predicate, and then identify the k pairs among them with the highest aggregate score. For this purpose, we could apply algorithms either like the R-tree join in [3], 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 in [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.

However, the above approaches do not provide a way to prioritize the join result computation according to the aggregate scores of the qualifying distance join pairs. Towards this direction, we present an optimized approach that employs aR-trees and manages to avoid computing the entire $R \bowtie_\epsilon S$ spatial ϵ-distance join. Algorithm 2 is a high-level pseudo-code of DFA. Different from SFA, DFA makes no pre-assumptions about the order of the objects in collection R and S. DFA also employs a min-heap C of size k to produce the final k-SDJoin results. In Lines 2–4, the algorithm computes the $R \bowtie_\epsilon S$ spatial distance join invoking the Join procedure. Join passes each (r, s) result pair to heap C, which keeps track of the k pairs with the highest aggregate score.

Procedure 2 is a pseudocode of the Join procedure. Given two aR-trees $\mathcal{A}_R$ and $\mathcal{A}_S$ (for input collections R and S, respectively), Join spatially joins the two trees by adapting the classic algorithm of [3] to traverse them not in a depth-first, but in a *best-first* order, which (i) still prunes entry pairs (e_R, e_S) with $e_R \in \mathcal{A}_R, e_S \in \mathcal{A}_S$ for which $dist(e_R, e_S) > \epsilon$ ($dist$ here denotes the minimum distance between the MBRs of the two entries), but (ii) it also prioritizes the entry pairs 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 and this order guarantees that the qualifying object pairs will be computed incrementally in decreasing order of their aggregate scores. To achieve this, Join employs a max-heap H which initially contains all pairs of root entries within distance ϵ from each other in the two trees (Lines 2–4). Pairs of entries from

Procedure 2: Join (for DFA)

Input : aR-trees $\mathcal{A}_R$, $\mathcal{A}_S$, spatial distance join threshold ϵ, monotone aggregate function γ, number of results k
Output : candidate set C
Variables : max-heap H of aR-tree entry pairs (e_R, e_S) organized by $\gamma(e_R, e_S)$, bound θ

1 **initialize** $\theta \leftarrow -\infty$;
2 **foreach** pair (e_R, e_S) in $\mathcal{A}_R.root \times \mathcal{A}_S.root$ **do** ▷ `Initialize heap` H
3 **if** $dist(e_R, e_S) \leq \epsilon$ **then**
4 H.push(e_R, e_S);
5 **while** $H \neq \emptyset$ **do**
6 $(e_R, e_S) \leftarrow H$.pop();
7 **if** $\gamma(e_R, e_S) \leq \theta$ **then** ▷ `Result secured`
8 **break**
9 **if** both e_R and e_S are non-leaf node entries **then**
10 $n_R \leftarrow$ node of $\mathcal{A}_R$ pointed by e_R;
11 $n_S \leftarrow$ node of $\mathcal{A}_S$ pointed by e_S;
12 **foreach** entry $e'_R \in n_R$ ***and*** each entry $e'_S \in n_S$ **do**
13 **if** $\gamma(e'_R, e'_S) > \theta$ ***and*** $dist(e'_R, e'_S) \leq \epsilon$ **then**
14 H.push(e'_R, e'_S);
15 **else**
16 **insert** (r, s) to C as next k-SDJoin result; ▷ `Update result`
17 **return** C;

H are examined (de-heaped) in priority of their aggregate scores $\gamma(e_R, e_S)$ as follows. The spatial distance join is evaluated for the corresponding aR-tree nodes and the results are inserted to H if they are non-leaf entries (branching condition at Line 9). Otherwise, if a leaf node entry pair (r, s) (i.e., an 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. Therefore, the object pair is included as the next result of the k-SDJoin query to the return set C (Lines 15-16). Finally, Join and thus DFA, terminates after k results have been computed.

Example 2 Consider the k-SDJoin query of Example 1. Initially, DFA builds the aR-trees of Figure 3. Next, DFA performs the aR-tree based spatial ϵ-distance join. The root nodes of the $\mathcal{A}_R$ and $\mathcal{A}_S$ aR-trees are first considered; (R_1, S_1), (R_2, S_2) entry pairs are added to max-heap H while the other two combinations (R_1, S_2), (R_2, S_1) are pruned by the ϵ-distance join predicate. The next pair to be examined is (R_1, S_1) as $\gamma(R_1, S_1) = 1.8 > \gamma(R_2, S_2) = 1.5$; the nodes pointed by R_1, S_1 are synchronously visited and their ϵ-distance join adds pairs (R_3, S_4), (R_4, S_4), (R_4, S_3) to H. 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 added to H. Then, (R_4, S_3) is de-heaped while (r_3, s_3) is added to H. The next pair to be popped from H is (r_3, s_3) which is guaranteed to be the ϵ-distance join pair with the highest aggregate score, since it is the first object pair extracted from max-heap H, and hence, DFA terminates as $k = 1$. ■

Correctness analysis. Similar to SFA, we prove the correctness of DFA by establishing the correctness of its two pruning criteria, i.e., Lines 3 and 7 in Procedure 2.

Lemma 3 *Given two aR-tree entries e_R and e_S, if $dist(e_R, e_S) > \epsilon$, every pair of objects in $e_R \times e_S$ can be safely pruned.*

Proof Based on the spatial property of aR-trees (inherited from R-trees), for every pair of objects $(r, s) \in e_R, \times e_S$, we have $dist(r, s) \geq dist(e_R, e_S)$ and consequently, $dist(r, s) > \epsilon$. Therefore, combining e_R and e_S cannot form any valid join results and the entry pair can be safely discarded. □

Lemma 4 *Given two aR-tree entries e_R and e_S, if $\gamma(e_R, e_S) \leq \theta$, every pair of objects in $e_R \times e_S$ can be safely pruned.*

Proof Based on the definition of the aR-tree, for every pair of objects $(r, s) \in e_R, \times e_S$, we have $r.score \leq e_R.score$ and $s.score \leq e_S.score$. Given a monotone function γ, we have $\gamma(r, s) \leq \gamma(e_R, e_S) \leq \theta$. Therefore, combining e_R and e_S cannot provide any join results with an aggregated score higher than the current k-th score θ, and (e_R, e_S) can be safely discarded. □

Last, we introduce the following theorem for the correctness of DFA:

Theorem 2 *DFA correctly computes k-*SDJoin*.*

Proof The theorem follows naturally from Lemmas 3 and 4. □

3 The Block-Based Algorithm

Due to primarily focusing on the ranking or the join sub-query/component of a k-SDJoin query, both SFA and DFA have particular shortcomings. SFA is expected to be fast only if the k join results are found after a few accesses over the sorted inputs R and S; otherwise, the overhead of repeatedly updating and probing aR-trees $\mathcal{A}_R$ and $\mathcal{A}_S$ can be too high. DFA is expected to be slower than SFA especially for large inputs and small k values. The entire spatial ϵ-distance join of the input collections will not be computed, but the algorithm still has to index all objects although part of them does not contribute to the k-SDJoin result.

3.1 Description

Similar to SFA, BA examines the objects in decreasing order of their scores. However, instead of probing *each* accessed *object* against the buffered objects from the other collection, BA each time probes a *block* of objects against the buffered *blocks* of objects. Moreover, before probing a new block of objects, BA creates an aR-tree for this block, and thus, the block-level probe corresponds to instances of DFA. Under this perspective, BA can be seen as an adaptation

ALGORITHM 3: Block-based Algorithm (BA)

Input : object collections R, S, spatial distance threshold ϵ, monotone aggregate function γ, number of results k
Output : result set $C = \{(r, s) \in R \times S : dist(r, s) \leq \epsilon\}$
Variables : aR-trees $\mathcal{A}_R$ and $\mathcal{A}_S$, thresholds θ and T, the lowest seen scores ℓ_R and ℓ_S

1 **initialize** $C \leftarrow \emptyset$, $\theta \leftarrow -\infty$, $\ell_R \leftarrow \infty$, $\ell_S \leftarrow \infty$;
2 **sort** R and S in descending order of the `score` attribute; ▷ `if not already sorted`
3 **while** more blocks of objects exist in R and S **do**
4 $i \leftarrow S$, **if** $\ell_S > \ell_R$; **otherwise** R ; ▷ `Determine current input`
5 $j \leftarrow R$, **if** $\ell_S > \ell_R$; **otherwise** S;
6 $b_i \leftarrow \mathsf{GetNextBlock}(i, \lambda)$; ▷ `Get next objects block from current input`
7 $\ell_i \leftarrow b_i^{\ell}$; ▷ `Update the lowest seen score from current input`
8 $\mathcal{A}_{b_i} \leftarrow \mathsf{CreateIndex}(b_i)$; ▷ `Bulk-load aR-tree for` b_i
9 **for** each block b_j of j **do**
10 **if** $\gamma(b_i{}^u, b_j{}^u) > \theta$ **then**
11 $\langle \theta, C \rangle \leftarrow \mathsf{Join}(\mathcal{A}_{b_i}, \mathcal{A}_{b_j}, T, \epsilon, \gamma, k, \theta, C)$; ▷ `Update current` C `and` θ
12 $T \leftarrow \max\{\gamma(h_R, \ell_S), \gamma(\ell_R, h_S)\}$; ▷ `Update termination threshold`
13 **if** $T \leq \theta$ **then** ▷ `Result secured`
14 **break**
15 **return** C

of DFA at the block level. BA also avoids computing the entire $R \bowtie_\epsilon S$ join employing the following bounds; each accessed block b is assigned a lower b^ℓ and an upper score bound b^u. Since the objects inside b are sorted in decreasing order of their score, b^u (b^ℓ) equals the score of the first (last) object in b.

Algorithm 3 is a pseudo-code of BA. First, in Lines 1–2, BA sorts inputs R, S (if needed, similar to SFA) and initializes min-heap C for the candidate/result pairs, bound θ and the lowest seen scores ℓ_R and ℓ_S from R and S, respectively. In Lines 3–14, BA evaluates the k-SDJoin query with an approach similar to SFA, but examining a block of objects at a time, instead of a single object. The next accessed collection and block are selected according to the highest last seen scores in Lines 4–6, similar to SFA; note that ℓ_R, ℓ_S are updated to upper score bound inside current block (Line 7). Without loss of generality, assume the next block b_i is accessed from collection R, i.e., $i = R$, $b_i = b_R$, $j = S$; the other case is symmetric. BA first constructs aR-tree $\mathcal{A}_{b_R}$ for current block and then joins b_R with every block accessed (buffered) so far from collection S. The blocks of S are considered in decreasing order of their score ranges (i.e., first b_{S_1}, then b_{S_2}, etc). A block pair (b_R, b_S) is joined similarly to a DFA call of the Join procedure but with two major differences. First, BA employs a novel block-based pruning technique based on the $\gamma(b_R^u, b_S^u)$ aggregate score which represents an upper score bound for all object pairs in $b_R \bowtie_\phi b_S$. If we have already found at least k candidates, i.e., $|C| = k$, then joining b_R with b_S is pointless when $\gamma(b_R^u, b_S^u) \leq \theta$ (recall that θ is the k-th highest aggregate score so far). In other words, current block b_R from R is joined only with blocks b_S from S for which $\gamma(b_R^u, b_S^u) > \theta$. Second, the Join procedure of BA updates min-heap C and bound θ similar to SFA. Finally,

PROCEDURE 3: Join (for BA)

Input : aR-trees $\mathcal{A}_R$ and $\mathcal{A}_S$, termination threshold T, spatial distance join threshold ϵ, monotone aggregate function γ, number of results k, k-th highest aggregate score θ, candidate set C
Output : updated bound θ, candidate set C
Variables : max-heap H of aR-tree entries, organized by aggregate scores

```
1  for each pair (e_R, e_S) in A_R.root × A_S.root do          ▷ Initialize heap H
2      if dist(e_R, e_S) ≤ ε then
3          H.push(e_R, e_S);
4  while H ≠ ∅ and T > θ do
5      (e_R, e_S) ← H.pop();
6      if γ(e_R, e_S) ≤ θ then
7          break
8      if e_R and e_S are non-leaf node entries then
9          n_R ← node of A_R pointed by e_R;
10         n_S ← node of A_S pointed by e_S;
11         for each entry e'_R ∈ n_R and each entry e'_S ∈ n_S do
12             if γ(e'_R, e'_S) > θ and dist(e'_R, e'_S) ≤ ε then
13                 H.push(e'_R, e'_S);
14     else
15         insert (r, s) to C, remove the k-th pair in C first if |C| = k;
16         θ ← aggregate score of the k-th pair in C;
17 return ⟨θ, C⟩;
```

block	id	loc	score
b_{R_1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R_2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R_3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R_4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

block	id	loc	score
b_{S_1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S_2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S_3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S_4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

Fig. 4 Example of BA with $\lambda = 2$ on the collections in Figure 2.

after handling current block, BA updates termination threshold T and checks the termination condition in Lines 12–14. Note that threshold T is the same as in SFA with h_R and h_S being the highest score from R and S, respectively, i.e., equal to the score of the very first object inside b_{R_1} and b_{S_1}.

Last, we elaborate on the Join procedure for BA. Procedure 3 illustrates the Join procedure of BA. Notice that, different from Procedure 2 and DFA, (i) the Join procedure for BA employs the termination threshold T in Line 4, and (ii) when the procedure identifies object pair (r, s) that qualifies the spatial ϵ-distance predicate with an aggregate score higher than bound θ, the pair is treated as a candidate result similar to SFA and thus, min-heap C and bound θ are updated accordingly in Lines 15–16.

Example 3 Consider the k-SDJoin query of Examples 1 and 2. We illustrate BA for $\lambda = 2$; Figure 4 shows the blocks to be accessed from each input collection. First, b_{R_1} is read and the $\mathcal{A}_{R_1}$ aR-tree is built. Then, b_{S_1} and b_{S_2} are accessed. The $\mathcal{A}_{S_1}$ and $\mathcal{A}_{S_2}$ aR-trees are built and joined with $\mathcal{A}_{R_1}$ producing no spatial join results. Next, the block b_{R_2} is read and joined with b_{S_1} and b_{S_2} (in this order), to generate $C = \{(r_3, s_3)\}$ and set $\theta = 1.6$. The next block b_{S_3} is joined with b_{R_1}, but not b_{R_2}, because $\gamma(b^u_{R_2}, b^u_{S_3}) = \gamma(0.8, 0.7) = 1.5 < \theta$; i.e., in the best case, a spatial distance join between b_{R_2} and b_{S_3} will produce a pair of aggregate score 1.5, which is not higher than the score of the current top pair (r_3, s_3). Next, the join between b_{S_3} and b_{R_1} does not improve current k-SDJoin result. At this point, BA terminates because the termination threshold $T = 1.5$ is not higher than $\theta = 1.6$, and $C = \{(r_3, s_3)\}$ is returned as the final result. ■

3.2 Correctness analysis

We first show the correctness of the block-based pruning criterion (Line 10 in Algorithm 3) and of Procedure 3; then, we prove the correctness of BA by contradiction.

Lemma 5 *Given blocks b_R and b_S, and the current candidate set C with k-th score θ, if $\gamma(b^u_R, b^u_S) \leq \theta$, every pair of objects in $b^u_R \times b^u_S$ can be safely pruned.*

Proof For every $(r, s) \in b_R \times b_S$, we have $r.score \leq b^u_R$ and $s.score \leq b^u_S$ which means that for the monotone function γ, $\gamma(r, s) \leq \gamma(b^u_R, b^u_S) \leq \theta$ also holds. Thus, combining b_R and b_S cannot provide join results with an aggregate score higher than θ. □

Lemma 6 *Procedure 3 determines all candidate results from a given block pair (b_R, b_S).*

Proof The two pruning criteria in DFA, $dist(e_R, e_S) > \epsilon$ and $\gamma(e_R, e_S) \leq \theta$, are also applied in Procedure 3 and we have already proved their correctness in Lemma 3 and Lemma 4, respectively.

To prove Lemma 6, we only need to prove that the new termination condition, $T \leq \theta$, is correct. This is straightforward because T is defined as the maximum possible aggregate score of any remaining (r, s) pairs, and θ is defined as the lowest score of the current result set C. Therefore, if $T \leq \theta$, it is impossible to find any better results and thus, Procedure 3 would never miss a candidate result pair. □

Theorem 3 *BA correctly computes k-SDJoin.*

Proof We proof the theorem by contradiction. First, assume that BA terminates after accessing blocks $b_i \in R$ and $b_j \in S$ with result set C, score bound θ (i.e., k-th score in C), $\ell_R = b^\ell_i$ and $\ell_S = b^\ell_j$. Second, assume that there is a join

combination (r, s) with $\gamma(r, s) > \theta$ not produced by BA; where r is contained in block b_m from R and s is inside block b_n from S.

The termination of BA implies that $\theta \geq T = \max\{\gamma(h_R, \ell_S), \gamma(\ell_R, h_S)\}$. So we have

$$\gamma(r, s) > \theta \geq \gamma(h_R, \ell_S) \tag{1}$$

and

$$\gamma(r, s) > \theta \geq \gamma(\ell_R, h_S) \tag{2}$$

Given Equation 1, as $r.score \leq h_R$, $s.score \geq \ell_S = b_j^\ell$ must hold. Further, since the input collections are sorted in descending order of the scores, $s.score \geq b_j^\ell$ with $s \in b_n$ imply that $n \leq j$. In a similar manner, we can also deduce $m \leq i$ from Equation 2. In other words, BA should have accessed blocks b_m and b_n before terminating, which means that BA did not produce (r, s) due to either of the following:

(1) Block pair (b_m, b_n) was discarded, which is disproved by the correctness of Lemma 5, or
(2) Procedure 3 missed object pair (r, s) while joining blocks b_m, b_n, which is also disproved by the correctness of Lemma 6.

Therefore, BA should have found (r, s), which contradicts the original assumption. □

3.3 Instance Optimality

We analyze the performance of BA based on the notion of *instance optimality* defined by Fagin et al. [8]. We first introduce the necessary notation and then establish BA's instance optimality building upon the analysis of [37] for relational rank joins.

Definition 1 (k-SDJoin Instance) *An instance of the k-SDJoin problem is an $(R, S, \epsilon, \gamma, k)$ tuple such that input collections R and S are accessed in decreasing order of their* `score` *attribute, ϵ is a spatial distance join threshold, γ is a monotone aggregate function, and $1 \leq k \leq |R \bowtie_\epsilon S|$.*

Note that Definition 1 defines a rank join instance of two inputs and one scoring attribute, similar to $I^{2\text{–}rel} \cap I^{1\text{–}dim}$ in [37], extended though to a spatial distance join predicate $dist(\cdot, \cdot) \leq \epsilon$.

Let $\mathbf{A}$ denote the class of *deterministic rank join* algorithms that solve a k-SDJoin instance with a behavior determined only by (i) the size of the input collections R and S, (ii) the objects $r \in R$ and $s \in S$ already examined by the algorithm, and (iii) the values of the aggregate function $\gamma(r, s)$ on pairs of these objects. In other words, an algorithm in $\mathbf{A}$ operates solely based on the knowledge it has from the objects accessed so far and it does not have any prior knowledge about the objects of the input collections. The cost of applying an algorithm $A \in \mathbf{A}$ is defined in terms of the number of objects accessed from each collection, denoted by $\texttt{topkdepth}(A, R)$ and $\texttt{topkdepth}(A, S)$.

Definition 2 (Access Cost) *Given a k-SDJoin instance $(R, S, \epsilon, \gamma, k)$ and an algorithm A, the* access cost *of A equals the total number of objects accessed from R and S:*

$$\texttt{acost}(A, R, S) = \texttt{topkdepth}(A, R) + \texttt{topkdepth}(A, S)$$

In [37], HRJN*/PBRJ$^*_c$ is proven to be *instance optimal* within the class of algorithms **A** with an optimality ratio of 2, for all instances of relational rank joins with equality join predicate. This finding can be straightforwardly extended for k-SDJoin and thus, SFA is also *instance optimal* within class **A** with an optimality ratio of 2, i.e., there exists a constant c, such that for any top-k join instance and any $A \in \mathbf{A}$:

$$\texttt{acost}(\text{SFA}, R, S) \leq 2 \cdot \texttt{acost}(A, R, S) + c$$

Finally, we establish the optimality of BA. Schnaitter and Polyzotis in [37] present a variant of the PBRJ framework where a *block* of objects, instead of a *single* object, at a time, is accessed from the input collections; in Section 7.2, we discuss how BA differs from previous block-based methods including the block variant of PBRJ. In this setup, the cost of applying a deterministic rank join algorithm A is defined with respect to the total number of accessed blocks.

Definition 3 (Block Access Cost) *Given a k-SDJoin instance $(R, S, \epsilon, \gamma, k)$ and an algorithm A, the* block access cost *of applying A equals the total number of λ sized blocks accessed from R and S:*

$$\texttt{bacost}(A, R, S) = \left\lceil \frac{\texttt{topkdepth}(A, R)}{\lambda} \right\rceil + \left\lceil \frac{\texttt{topkdepth}(A, S)}{\lambda} \right\rceil$$

The instance optimality analysis conducted in [37] for HRJN*/PBRJ$^*_c$, and thus, also for SFA in case of k-SDJoin, can be directly extended to the `bacost` metric in place of `acost` since the block variant of SFA processes an object similar to the original method. In practice, BA extends and optimizes the block variant of SFA with the purpose of reducing the computational cost of k-SDJoin while employing the same bound scheme to determine termination threshold T and the same pulling strategy. Thus, similar to the block variant of SFA, BA is instance optimal within the class of deterministic ranked join algorithms **A** for all k-SDJoin instances of Definition 1.

3.4 Discussion

Complexity. Evaluating k-SDJoin queries with BA is dominated by the tasks of indexing blocks and performing block-joins. As shown in Section 6.2, the accessing cost is several times lower than the computational cost, and so our complexity discussion here focuses on the indexing and the block-join costs. [5]

[5] We briefly discuss the cost of automatically determining block size λ in the next section.

In the worst case, BA needs to index all $\frac{|R|}{\lambda}$ blocks from input R and all $\frac{|S|}{\lambda}$ from S. The indexing cost for a block is dominated by the $\mathcal{O}(\lambda \cdot log\lambda)$ cost for sorting its contents; hence, BA spends $\mathcal{O}((|R| + |S|) \cdot log\lambda)$ time for indexing, overall. On the other hand, in the worst case when no pruning is effective, BA has to evaluate every possible block-join, i.e., $\frac{|R|}{\lambda} \cdot \frac{|S|}{\lambda}$ block-joins in total, each of which costs $\mathcal{O}(\lambda^2)$. Overall, BA spends $\mathcal{O}(|R| \cdot |S|)$ time for joining.

Comparison to SFA. Despite the resemblance, BA has two major advantages over SFA. First, performing joins at the block level is more efficient because (i) each block of objects is indexed only once and efficiently, in a bulk-loading manner, instead of iteratively inserting objects as in SFA, and (ii) the aR-tree of a given block can be used for multiple block-level joins. Second, in BA, the current block is joined only with a small number of blocks from the other collection (thanks to the block bound based pruning), while in SFA the current object is probed against all buffered objects. This is critical as the costs of maintaining and querying an aR-tree depend on its size.

Multiple Scoring Attributes per Input. In this case, the objects of each input, e.g., $r \in R$, are accessed in order of their aggregate score bound $\overline{\gamma}(r)$, defined by applying γ on r's scoring attributes while setting the attributes of input S to their maximum value. BA can be directly applied in this setup, but it is no longer instance optimal similar to the case of relational rank joins studied in [37]. This is because of using the corner bounding scheme for termination threshold T (similar to HRJN*/PBRJ$^*_c$). BA extends and optimizes PBRJ for k-SDJoin in an orthogonal direction and hence, we can employ the FR^* bound scheme for determining a tight threshold T and the PA pulling strategy of the FRPA algorithm from [10] to provide both an instance optimal and a computationally efficient evaluation method for k-SDJoin with multiple scoring attributes. Note that FRPA corresponds to the PBRJ$^{PA}_{FR^*}$ specialization of the PBRJ framework.

4 Models

An issue that remains open is how to determine an appropriate block size λ for BA. Intuitively, there exists a trade-off between the response time of BA and its block size: small λ values incur a high overhead in the indexing cost, and hence, BA benefits less from the block-wise join evaluation; while with a large λ, BA resembles an improved but still inefficient version of DFA, which over-computes unnecessary part of the spatial ϵ-distance join $R \bowtie_\epsilon S$. Selecting λ for BA is an important yet challenging task, which is highly dependent on the employed indexing structures, data distribution, etc. In Section 4.1, we model automatic block size tuning as an optimization problem. Then, in Section 4.2, we propose a novel model for estimating the number of objects to be accessed from each input collection which as shown in Section 6 enhances the accuracy of our model for selecting block size λ.

4.1 Selecting Block Size for BA

The optimal block size minimizes the computational cost of BA, captured by the *objective cost function*:

$$\mathcal{C}(\lambda) = |N_{index}(\lambda)| \cdot \mathcal{C}_{index}(\lambda) + |N_{join}(\lambda)| \cdot \mathcal{C}_{join}(\lambda) \tag{3}$$

Intuitively, the $|N_{index}(\lambda)| \cdot \mathcal{C}_{index}(\lambda)$ part of the $\mathcal{C}(\lambda)$ objective function equals the overall *indexing cost* for BA with $|N_{index}(\lambda)|$ being the total number of indexed blocks from the input collections, and $\mathcal{C}_{index}(\lambda)$ being the cost of indexing a block. The $|N_{join}(\lambda)| \cdot \mathcal{C}_{join}(\lambda)$ part equals the overall *joining cost* for BA with $|N_{join}(\lambda)|$ being the total number of block-level joins performed, and $\mathcal{C}_{join}(\lambda)$ being the cost of joining two blocks. In what follows, we elaborate on each factor of the $\mathcal{C}(\lambda)$ objective function and then discuss how the value of λ that minimizes $\mathcal{C}(\lambda)$, can be estimated.

Regarding $\mathcal{C}_{index}(\lambda)$ and $\mathcal{C}_{join}(\lambda)$, both costs are determined by how a block-level join is implemented. Under the aR-tree based evaluation of BA, the indexing cost is dominated by the cost of sorting a block to bulk-load its aR-tree, i.e., $\mathcal{C}_{index}(\lambda) = \alpha_1 \cdot \lambda \cdot \log \lambda + \alpha_2$, while the joining cost for two aR-trees is quadratic to the block size λ, i.e., $\mathcal{C}_{join}(\lambda) = \alpha_3 \cdot \lambda^2 + \alpha_4$. In Section 6, we derive α_1, α_2, α_3, α_4 by conducting a series of tests varying λ, and then, employing regression analysis over the collected $(\lambda, \mathcal{C}_{index}(\lambda))$ and $(\lambda, \mathcal{C}_{join}(\lambda))$ values.

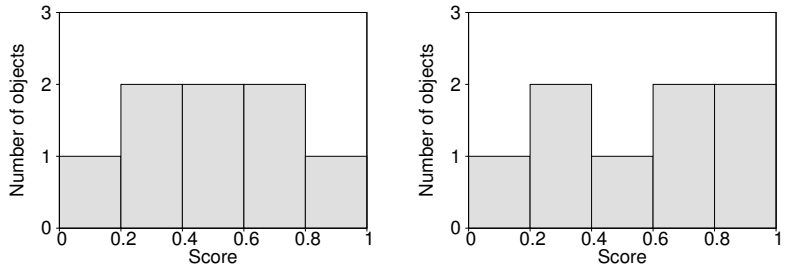
Next, we consider $|N_{index}(\lambda)|$. Let d_R and d_S be the total number of objects examined from each input by SFA (also known as the *top-k depth* for collections R and S). As BA and SFA employ the same bounding scheme to define the termination threshold T, BA terminates after accessing blocks $b_{R_{\lceil d_R/\lambda \rceil}}$ and $b_{S_{\lceil d_S/\lambda \rceil}}$. Hence, BA will index in total

$$|N_{index}(\lambda)| = \lceil d_R/\lambda \rceil + \lceil d_S/\lambda \rceil \tag{4}$$

blocks. Last, to determine the total number $|N_{join}(\lambda)|$ of block-level joins by BA we adopt *any-k depth* c_R, c_S introduced in [15] as the minimum number of objects examined from each input to identify the first k candidate pairs. These first k pairs are not necessarily among the final k-SDJoin results, and by definition $c_R \leq d_R$ and $c_S \leq d_S$. With any-k depths c_R and c_S, the execution of BA is divided in two parts.

First, until blocks $b_{R_{\lceil c_R/\lambda \rceil}}$ and $b_{S_{\lceil c_S/\lambda \rceil}}$ are accessed, BA cannot employ the $\gamma(b^u_{R_i}, b^u_{S_j}) \leq \theta$ pruning because less than k candidate pairs are found and threshold θ (i.e., the aggregate score of the current k-th best candidate is not defined. Thus, all $\lceil c_R/\lambda \rceil \cdot \lceil c_S/\lambda \rceil$ block-level joins are to be computed. Second, after blocks $b_{R_{\lceil c_R/\lambda \rceil}}$ and $b_{S_{\lceil c_S/\lambda \rceil}}$ are accessed, threshold $\theta = \gamma(r_{c_R}, s_{c_S})$ and the $\gamma(b^u_{R_i}, b^u_{S_j}) \leq \theta$ pruning can be applied.[6] To approximate the total number of block-level joins that qualify the pruning criterion, denoted by $|N^{c_R,c_S \to d_R,d_S}_{join}(\lambda)|$, we need to estimate (i) the score of the r_{c_R}, s_{c_S} objects

[6] We denote by r_{c_R} and s_{c_S} the c_R and c_S-th objects in the sorted inputs, respectively.



(a) $\mathcal{H}_R$ (b) $\mathcal{H}_S$

Fig. 5 Score histograms for the collections in Figure 2.

to define threshold θ, and (ii) the upper score bound of every block, i.e., the score of its very first object in the block. Then, without loss of generality, fix a block b_R from collection R. We consider every b_S block of S in between $b_{\lceil s_{c_S}\lambda\rceil}$ and $b_{\lceil d_S/\lambda\rceil}$; $|N_{join}^{c_R,c_S\rightarrow d_R,d_S}(\lambda)|$ is increased by one for each block b_S with $\gamma(b_R^u, b_S^u) > \theta$ denoting that BA will perform the $b_R \bowtie_\epsilon b_S$ spatial ϵ-distance block-level join.

Last, to address points (i) and (ii), we employ histogram statistics from the inputs; equi-width histograms $\mathcal{H}_R$ and $\mathcal{H}_S$ summarize the *score* distributions in collections R and S, respectively. Unlike the expensive multi-dimensional histograms in [7,38], 1-dimensional $\mathcal{H}_R$, $\mathcal{H}_S$ can be derived with low cost from the collections based on an independence assumption or via sampling. Note that in case the inputs are not pre-sorted on their `score` attribute (as required by BA), histograms $\mathcal{H}_R$ and $\mathcal{H}_S$ can be computed during the sorting process. Figure 5 illustrates $\mathcal{H}_R$ and $\mathcal{H}_S$ score histograms for the collections in Figure 2. A single pass over $\mathcal{H}_R$ suffices to identify which interval contains the score of an object r_i. Then, without loss of generality, we set the score of r_i to the upper bound of the score interval.

Example 4 Consider for instance $\mathcal{H}_R$ in Figure 5(a) and assume that we want to estimate the upper score bound of block b_{R_2} in Figure 4 with $\lambda = 2$. By definition, $b_{R_2}^u$ equals the score of the first object in b_{R_2}, i.e., r_3. The objects in R are sorted in decreasing order of their scoring attribute, and so, r_3 is the 3rd object of collection R. As score intervals $(0.8, 1.0]$ *and* $(0,6, 0.8]$ *in $\mathcal{H}_R$ contain 1 and 2 objects, respectively, the score of r_3 should fall inside* $(0,6, 0.8]$ *which gives an estimation of* 0.8. ■

Summing up, BA will perform

$$|N_{join}(\lambda)| = \lceil c_R/\lambda\rceil \cdot \lceil c_S/\lambda\rceil + |N_{join}^{c_R,c_S\rightarrow d_R,d_S}(\lambda)| \quad (5)$$

block-level joins, in total.

With the objective cost function $\mathcal{C}(\lambda)$ defined using Equations (3)–(5), we now focus on estimating the λ value that minimizes $\mathcal{C}(\lambda)$. Following the definition of top-k depth, a block should contain at most $max\{d_R, d_S\}$ objects since no more than d_R, d_S objects are examined from the collections. To efficiently determine a good value for λ inside $[1, max\{d_R, d_S\}]$, we employ the

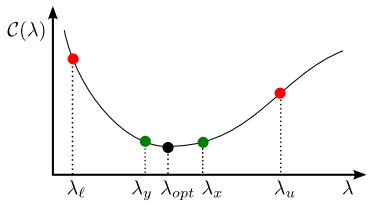


Fig. 6 Example of the golden section search.

golden section search technique [17]. The golden section search is an efficient way to progressively narrow the range that contains the minimum of a function. Figure 6 exemplifies the procedure of determining the minimum λ_{opt} for $\mathcal{C}(\lambda)$, operating in a divide-and-conquer manner; note that λ_{opt} is initially contained inside interval $[1, max\{d_R, d_S\}]$. At each follow up iteration, the search considers a triple of λ values; the current $[\lambda_\ell, \lambda_u]$ interval which contains λ_{opt} and a split point $\lambda_x \in [\lambda_\ell, \lambda_u]$, determined such that the distances between values λ_ℓ, λ_x and λ_u follow the golden ratio (hence the name of the method), i.e., $\frac{\lambda_u - \lambda_x}{\lambda_x - \lambda_\ell} = \phi = \frac{1+\sqrt{5}}{2}$. We then determine a second split point λ_y (again following the golden ratio) which according to the search strategy should lie inside the largest between $[\lambda_\ell, \lambda_x]$ and $[\lambda_x, \lambda_u]$; in our example, $[\lambda_\ell, \lambda_x]$. Since $\mathcal{C}(\lambda_y) < \mathcal{C}(\lambda_\ell)$, the next iteration of the search should consider interval $[\lambda_\ell, \lambda_x]$ and λ_y as the split point. The search terminates when $|\lambda_x - \lambda_y|$ is smaller than a preset threshold, and $\lambda_{est} = \frac{\lambda_x + \lambda_y}{2}$ is returned as the estimated λ. The time complexity of the golden section search is $\mathcal{O}(log(max\{d_R, d_S\}))$; in practice, our experiments have clearly shown that the overall cost of determining block size λ is negligible compared to the indexing and joining costs involved in BA.

4.2 Estimating any-k and top-k Depths

Next, we discuss how depths c_R, d_R, c_S, d_S are estimated. One option is to adopt the model proposed in [15] for relational rank joins (note that in the absence of multi-dimensional statistics, the model in [38] operates similar to [15]). Specifically, assuming that $c_R \cdot c_S \cdot \sigma \approx k$, any-$k$ depths are set to $c_R = c_S = \sqrt{k/\sigma}$, where σ is the join selectivity of the input collections (computed via sampling), and top-k depths are set to $d_R = d_S = 2 \cdot c_R = 2 \cdot \sqrt{k/\sigma}$. This model however is of limited applicability; both the join and the scoring attribute need to follow a *uniform distribution* while there should exist *no correlation* between these two attributes. First, with a uniformly distributed join attribute a good estimation for join selectivity σ is achieved via sampling and thus, also a good estimation of any-k depths c_R, c_S. Second, a uniformly distributed scoring attribute is required for estimating d_R, d_S as twice the any-k depth. In practice, however, either of these assumptions may not hold for k-SDJoin. For instance, as pointed out even in [15], in a hierarchy of joins where the output of one top-k join operator serves as input to another, the score

distributions of higher level joins tend to be normal. Formulas for computing c_R, c_S, d_R, d_S for the high level joins were provided, but the model in [15] still requires knowledge of the hierarchy and the distribution of both join and scoring attributes. Further, our analysis in Section 6 on real collections showed that a correlation between the spatial location and the scoring attribute may exist. For example hotels located close to important landmarks are usually assigned higher scores compared to the rest. Last, this model sets $c_R = c_S$ and $d_R = d_S$, which in practice, rarely holds even if $|R| = |S|$.

To overcome these limitations, we devise a novel model for any-k and top-k depths which uses cheap-to-compute and maintain statistics, and is able to better cope with the special characteristics of the inputs. In Section 6, we show the superiority of our model over [15]. We first discuss any-k depths. Our analysis on real data showed that c_R, c_S are determined by the join selectivity for the upper part of the sorted inputs, i.e., the highly scored objects, and usually differs from the join selectivity of the entire collections. Hence, the idea behind our approach is to repeatedly sample the upper part of the input collections and compute their join selectivity until for the potential number of join results denoted by k', $k \leq k' \leq \delta \cdot k$ holds (k is the number of desired results for top-k join and δ is a tuning parameter to avoid $k' \gg k$). We initially focus on the first t_R and t_S objects from R and S, respectively, and estimate their join selectivity σ_t via sampling. To initialize tuning parameters t_R, t_S, we use the following simple heuristic; the initial values are set such that the $t_R : t_S$ ratio follows the $|R| : |S|$ ratio for the input cardinalities. Based on the potential number of results $k' = t_R \cdot t_S \cdot \sigma_t$, we distinguish between two cases. If $k' < k$, we need to join a larger part of the inputs to find the first k candidate pairs, i.e., $c_R > t_R$ and $c_S > t_S$, and so, we repeat the process increasing t_R, t_S by ξ^+ times. Otherwise if $k' > \delta \cdot k$, although we have enough join results, we repeat the process decreasing t_R, t_S by ξ^- times to better estimate c_R, c_S (tuning parameters ξ^+, ξ^- are selected such that $\xi^+ > 1 > \xi^- > 0$). The above process terminates when $k \leq k' \leq \delta \cdot k$, and c_R, c_S are set to current t_R, t_S, respectively. Regarding the initial values t_R, t_S, even when $c_R : c_S \neq |R| : |S|$ holds, our simple initialization heuristic still allows our model to converge to different values of c_R, c_S which are very close to the real values, as shown in Section 6. Note that the problem of estimating the join selectivity is essentially orthogonal to our study; for example methods such as [2,9] can be used, but for matters of simplicity and generality we choose to employ sampling. Overall, the cost of estimating any-k depths c_R, c_S is $\mathcal{O}(n \cdot \psi^2 \cdot c_R \cdot c_S)$, where ψ is our sampling ratio and n is the number of joins computed on samples.

Finally, to estimate top-k depths d_R, d_S, we employ threshold $\theta = \gamma(r_{c_R}, s_{c_S})$ defined after accessing blocks $b_{R_{\lceil c_R/\lambda \rceil}}$ and $b_{S_{\lceil c_S/\lambda \rceil}}$. Specifically, d_R equals the number of objects in input R whose aggregate score γ with the highest scored object in S, i.e., s_1, is higher than θ:

$$d_R = |\{r_i \in R \colon \gamma(r_i, s_1) > \theta\}|, \; d_S = |\{s_j \in S \colon \gamma(r_1, s_j) > \theta\}| \qquad (6)$$

To evaluate Equation (4) we employ again the $\mathcal{H}_R$ and $\mathcal{H}_S$ score histograms. For simplicity, we focus on d_R. The goal is to find the last record r_i in R with

$\gamma(r_i, s_1) > \theta$. Given threshold $\theta = \gamma(r_{c_R}, s_{c_S})$ and the score of s_1, we first deduce a lower bound ψ_R for the score of such an object r_i, and then identify the score interval of $\mathcal{H}_R$ that contains ψ_R. With this interval, we also get how many intervals involve scores higher than ψ_R and hence, how many objects in collection R have a score higher than ψ_R, i.e., top-k depth d_R, as the aggregate score of these objects with s_1 is by definition higher than θ. Overall, the cost of estimating top-k depths d_R, d_S is linear to the number of buckets inside score histograms $\mathcal{H}_R$, $\mathcal{H}_S$.

Example 5 Consider Figure 5 and assume that we have already estimated any-k depths $c_R = c_S = 2$. With monotone aggregate function $\gamma = SUM$, we have threshold $\theta = \gamma(r_{c_R}, s_{c_S}) = SUM(0.8, 0.8) = 1.6$. Hence, in order for an object r_i to have an aggregate score with s_1 higher than θ, i.e., $\gamma(r_i, s_1) = \gamma(r_i, 0.9) > 1.6$, its score needs to be higher than $\psi_R = 0.7$. According to $\mathcal{H}_R$, score bound ψ_R falls inside interval $(0.6, 0.8]$. Assuming again that all objects whose score falls inside $(0.6, 0.8]$ have a score equal to the upper bound 0.8, collection R contains approximately 3 objects with score higher than ψ_R, i.e., 2 from interval $(0.6, 0.8]$ and 1 from $(0.8, 1.0]$, and thus, we set $d_R = 3$. ■

5 The Case of Multiple Inputs

In the previous sections, we focused solely on a binary k-SDJoin operator. We next discuss how BA handles the case of multiple input collections; note that similar ideas can be employed for SFA, and DFA in the presence of special optimizations which allow us to incrementally produce join results. Essentially, there exist two approaches for evaluating a k-SDJoin with more than two inputs. The first approach is to treat all inputs in a multi-way join fashion (similar to relational rank joins discussed in [37]); a multi-way variant of BA accesses the next block b_i of λ objects from input R_i with $\gamma(\{b^u_{R_j} | j \neq i\}, b^\ell_{R_i}) > \max_{j \neq i} \gamma(b^\ell_{R_j}, \{b^u_{R_{j'}} | j' \neq j\})$, and joins b_i with the buffered blocks from all other inputs. The second approach (similar to [14,20]) is to form a hierarchy of k-SDJoin binary operators; in what follows, we elaborate on this approach.

Without loss of generality, we assume that two k-SDJoin operators are applied on three inputs R_1, R_2 and R_3; Figure 7 presents an execution plan for this query. The evaluation is driven by the top level k-SDJoin operator which incrementally joins R_3 with the intermediate results of $R_{1,2} = R_1 \bowtie_\epsilon R_2$, i.e., object pairs from $R_{1,2}$ are generated on-demand in decreasing order of their aggregate score and pipelined to the top level operator. In practice, the k-SDJoin between R_1 and R_2 is implemented as a `GetNextBlock` iterator which pipelines the next λ object pairs. For the top-level k-SDJoin between $R_{1,2}$ with R_3 we can directly apply Algorithm 3 and implement Join as in Procedure 3, but for joining R_1 with R_2, we need to define an incremental variant of BA (for simplicity denoted by *inc*BA). Essentially, *inc*BA progressively produces the results of the $R_1 \bowtie_\epsilon R_2$ join sorted by their aggregate score, in blocks of λ objects; note that in practice, we do not need to fully compute $R_1 \bowtie_\epsilon R_2$

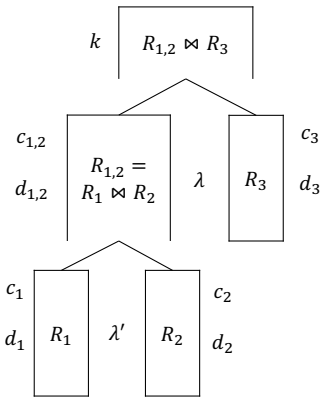


Fig. 7 Example of a 2-level execution plan for k-SDJoin on collections R_1, R_2, R_3.

due to the termination condition of the top-level k-SDJoin operator which consumes the $R_{1,2}$ blocks. In this spirit, *inc*BA defines and employs the $T = max\{\gamma(h_R, \ell_S), \gamma(\ell_R, h_S)\}$ termination threshold of Lines 12–14, Algorithm 3 in order to stop after producing the next λ objects, but cannot exploit the $\gamma(b_i^u, b_j^u) > \theta$ block-join pruning (Line 10, Algorithm 3) and the $\gamma(e'_R, e'_S) > \theta$ or $dist_L(e'_R, e'_S) > \epsilon$ aR-tree based pruning (Line 10, Procedure 3) in order to ensure correctness.

A second challenge relates to the automatic selection of the block size for the involved k-SDJoin operators. Consider again the query with inputs R_1, R_2 and R_3 and k-SDJoin's execution plan in Figure 7. As discussed in Sections 4.1 and 4.2, the process of selecting an appropriate block size relies on estimating any-k and top-k depths from the input collections and hence also on the number of required results k and on sampling the upper part of sorted inputs. Under this, our model from Section 4 is not directly applicable in either of the k-SDJoin operators of Figure 7. In particular, for the top-level operator, the object pairs from $R_{1,2}$ are incrementally produced which means that the $\mathcal{H}_{1,2}$ histogram is not available in advance and sampling is limited. On the other hand, for the bottom-level k-SDJoin, the number of required results is unknown as the `GetNextBlock` iterator may be called multiple times. But, we can address the above issues in a progressive manner, as follows.

Initially, we set the λ block size for the top-level k-SDJoin equal to a predefined tuning parameter λ_0. This initialization will allow us to call the `GetNextBlock` iterator of the bottom-level k-SDJoin for the first time. Intuitively, the *inc*BA will evaluate a λ_0-SDJoin with R_1 and R_2 as inputs, thus our model for binary k-SDJoin will now be able to automatically select a λ' block size for this operator. With the first λ_0 object pairs pipelined to BA of the top-level k-SDJoin, we can now sample R_3 and current $R_{1,2}$, compute any-k depths c_3, $c_{1,2}$ and top-k depths d_3, $d_{1,2}$, and appropriately update λ. Regarding the required $\mathcal{H}_{1,2}$ histogram we identify two options; we can either combine $\mathcal{H}_1$ and $\mathcal{H}_2$ with respect to the aggregate function γ, e.g., as in [19], or

build it using current version of $R_{1,2}$; this issue is left for future investigation. Finally, at the top-level k-SDJoin, after BA finishes examining the first block of λ_0 object pairs, a new call on the `GetNextBlock` iterator of the bottom-level k-SDJoin is issued demanding the next λ results from $R_{1,2}$, i.e., based on the updated block size for the top-level k-SDJoin. This process is repeated until the termination condition on the top-level for the k required results is met. Note that at each round, our depth estimation and block size selection models will be able to improve $c_1, c_2, d_1, d_2, c_{1,2}, d_{1,2}, c_3, d_3$ and λ', λ, respectively, and hence, accelerate the overall evaluation of the query.

6 Experimental Evaluation

In this sections, we present an experimental evaluation of our techniques for k-SDJoin. Section 6.1 details the setup of our analysis. Section 6.2 justifies our focus on the efficient, in terms of CPU cost, k-SDJoin evaluation while Section 6.3 our decision to use aR-tree as the indexing structure. Sections 6.4 and 6.5 demonstrate the effectiveness of our models for any-k / top-k depths, and for selecting block size λ in BA, respectively. Section 6.6 conducts an extensive comparison of the SFA, DFA and BA algorithms. All methods were implemented in C++ and the tests run on a 2.3 GHz Intel Core i7 CPU with 8GB of RAM.

6.1 Experimental Setup

Our analysis involves both real-world and synthetic datasets. Regarding the former, we used a collection of 645K hotels from Booking.com denoted by HOTELS, and a set of 481K restaurants from TripAdvisor.com denoted by RESTS. Join attribute `loc` stores the location of an object (hotel or restaurant) in the 2-dimensional space and attribute `score` is the average user rating. For the rest of this analysis, we denote this test by HOTELS-RESTS. We also used collections of real spatial locations with synthetic scores. Specifically, FLICKR contains 1.7M locations associated with photographs in the city of London from Flickr.com, while ISLES is a collection of 20M POIs in the area of the British Isles from OpenStreetMap.org. For our tests, each of the above collections (with generated scores) is split into two disjoint partitions to avoid result pairs involving the same object.

To generate scores for FLICKR and ISLES, we analyzed the real-world datasets resulting in two observations. First, object scores usually follow a normal distribution, as shown in Figure 8(a) for hotels in Paris from TripAdvisor.com. Second, a correlation between join attributes and scores may exist. In Figure 8(b), the rating of a hotel is denoted by how dark its red marker is. Most of the highly rated hotels are close to each other, and conveniently located next to an important landmark of Paris, the River Seine. Under these observations, we distinguish between attribute `score` of type IND and CORR;

Table 2 Experimental parameters.

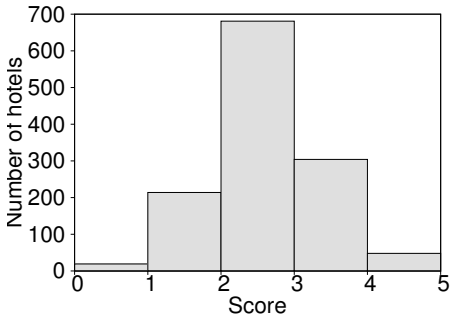
description	parameter	values	default value
Join selectivity	ϵ	$0.0001, 0.0005, 0.001, 0.005, 0.01$	0.001
Number of results	k	$1, 5, 10, 50, 100$	10
Number of seeds (CORR)	$\|\Sigma\|$	$10, 20, 50, 100$	20
Number of objects ($\times$1,000,000)	$\|R\| + \|S\|$	2.5, 5, 10, 20	10
Cardinality ratio	$\|R\| : \|S\|$	1, 2, 3, 4, 5	1

similar to [41]. For IND, `score` values are normally distributed inside the $[0, 1]$ interval and independent to the values of the join attribute `loc`. In contrast, for CORR, we first randomly generate $|\Sigma|$ *seeds*, and assign to each of them a score uniformly distributed inside the $[0, 0.8]$ interval. The generated objects are divided into $|\Sigma|$ clusters based on their spatial distance to the seeds and the score of each object equals the score of its closest seed plus a noise normally distributed inside $[0, 0.2]$.

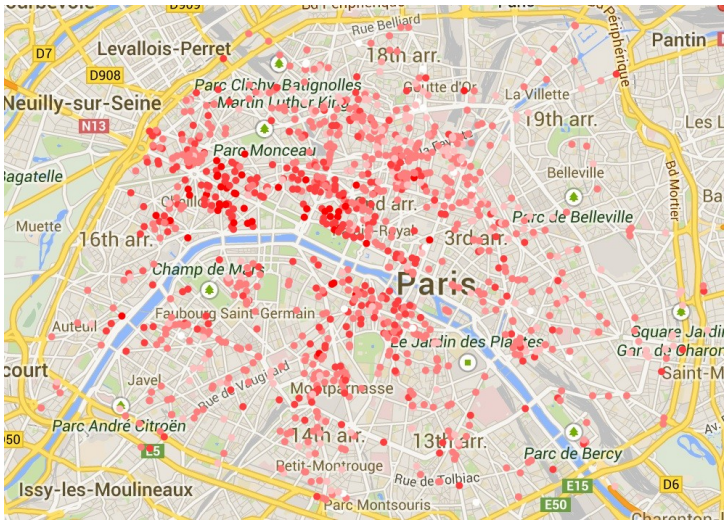
To assess the performance of SFA, DFA and BA, we measure their response time for $\gamma = SUM$ (note that our analysis can be directly extended to any monotone aggregate function), including any indexing and/or sorting costs, while varying: (i) the join selectivity, captured by distance threshold ϵ, (ii) the number of requested results k, and (iii) the number of seeds $|\Sigma|$ for synthetic collections of CORR scores. We also perform scalability and cardinality tests over subsets of the synthetic collections varying parameters $|R| + |S|$ and $|R| : |S|$. Table 2 summarizes all parameters involved in our study. On each test, we vary one parameter; the rest are set to their default value. Note that as the value of $|\Sigma|$ increases the score generator produces more independent and less correlated scores; for $|\Sigma| = 100$, the generated scores are uniformly distributed. Last, also note that both the input collections and their created aR-trees (with a 4KB page size) are stored in main memory. In Section 6.7, we discuss how our analysis can be extended when all involved data do not entirely fit inside the available main memory.

6.2 Focus on Computational Cost

We first justify our decision to focus on CPU efficient k-SDJoin evaluation, instead of minimizing the object accesses from the input collections, which has been the primary target of previous studies for relational rank joins. For this purpose, we assume that inputs R and S reside on disk, already sorted in decreasing order of their scoring attribute `score`, and SFA gradually accesses their objects to evaluate k-SDJoin queries. To calculate the number of I/Os incurred by SFA, we consider a 16KB page (typical for modern database systems) and the worst case scenario of performing only random accesses. To measure the access cost of SFA, we charge 10ms for each page access (corresponding to a 7200rpm HDD). Table 3 reports the number of I/Os, the access



(a) Score distribution



(b) Hotels' location and score

Fig. 8 Hotels in Paris from TripAdvisor.com.

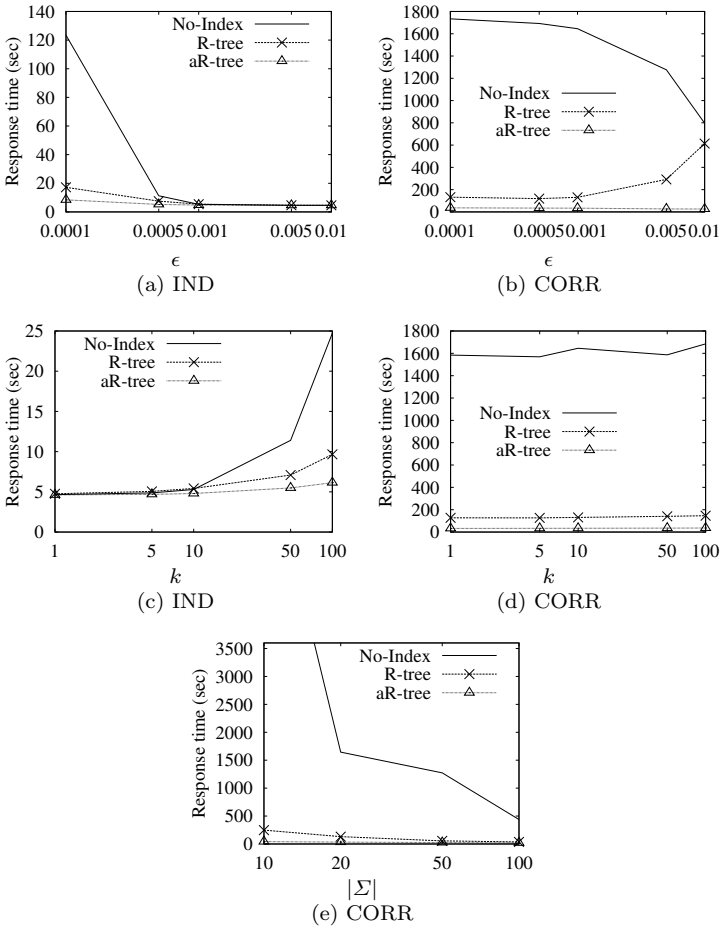
and the CPU cost of SFA to compute k-SDJoin. The test clearly shows that CPU cost overshadows access cost, being up to an order of magnitude higher; an exception arises when the number of accessed objects is small, e.g., FLICKR with IND scores. Note that in practice access cost can be even less important as some pages are sequentially accessed (e.g. data prefetching [40]) or modern disk hardware (e.g., SSDs) is used.

6.3 Selecting Index Structure for k-SDJoin

As discussed in Sections 2 and 3, although R-tree is the dominant indexing structure for spatial data, SFA, DFA and BA, all employ the aR-tree for computing k-SDJoin. In what follows, we justify this decision by comparing three alternative implementations for the evaluation algorithms, termed:

Table 3 Access and CPU cost of SFA (for default k, ϵ).

collection	score type	number of I/Os	access cost (seconds)	CPU cost (seconds)
HOTELS-RESTS	–	44	0.44	1.48
FLICKR	IND	2	0.02	0.02
	CORR	128	1.28	3.13
ISLES	IND	9	0.09	0.19
	CORR	704	7.04	29.16



(a) IND (b) CORR (c) IND (d) CORR (e) CORR

Fig. 9 Comparison of SFA alternatives: scoring attributes of type IND & CORR and ISLES.

1. the aR-tree, presented in Sections 2 and 3;
2. the R-tree, which uses R-trees instead of aR-trees to index the spatial join attribute `loc`.

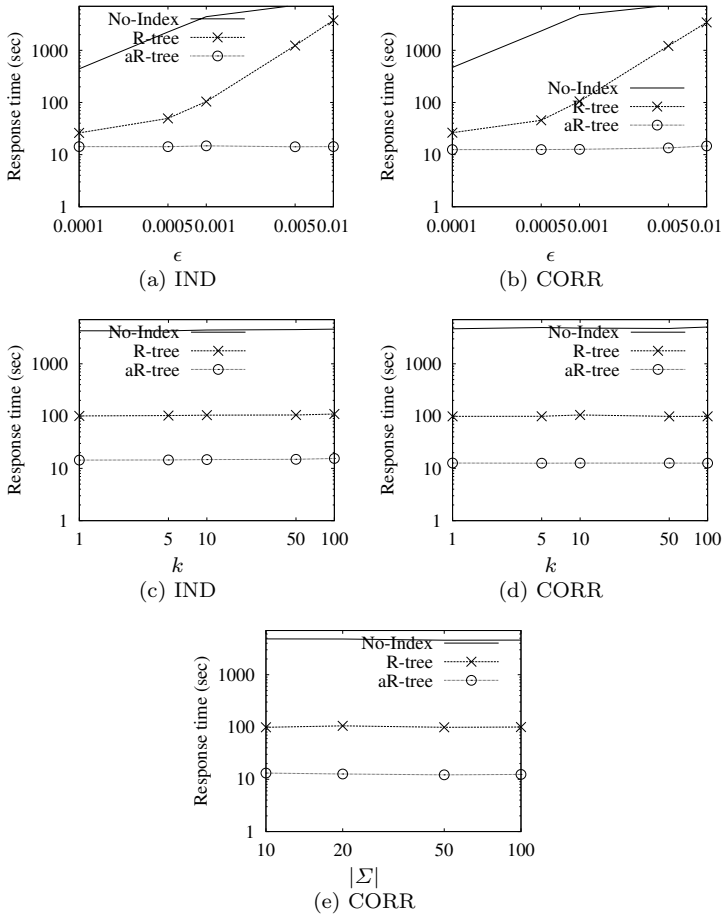


Fig. 10 Comparison of DFA alternatives: scoring attributes of type IND & CORR and ISLES (Log-Scaled).

3. the No-Index, which performs a scan against the buffered objects in case of SFA, while sorts the objects and applies a spatial plane sweep join technique in case of DFA and BA.

Figure 9 reports the response time of SFA alternatives in case of IND and CORR scoring attributes on ISLES, while varying the ϵ, k and $|\Sigma|$ parameters. We observe that the aR-tree alternative outperforms the other two as it is able to prune object pairs in terms of both their spatial distance and their aggregate scores. We also observe that the behaviour of the R-tree alternative differs when increasing ϵ on IND or CORR scoring types. The reason is the following. As ϵ increases, more object pairs qualify the spatial ϵ-distance join predicate. Since the objects are sorted in descending order of their scores, a smaller number of pairs needs to be examined. However, the increase of ϵ also incurs a higher cost for the range queries performed by the R-tree alternative.

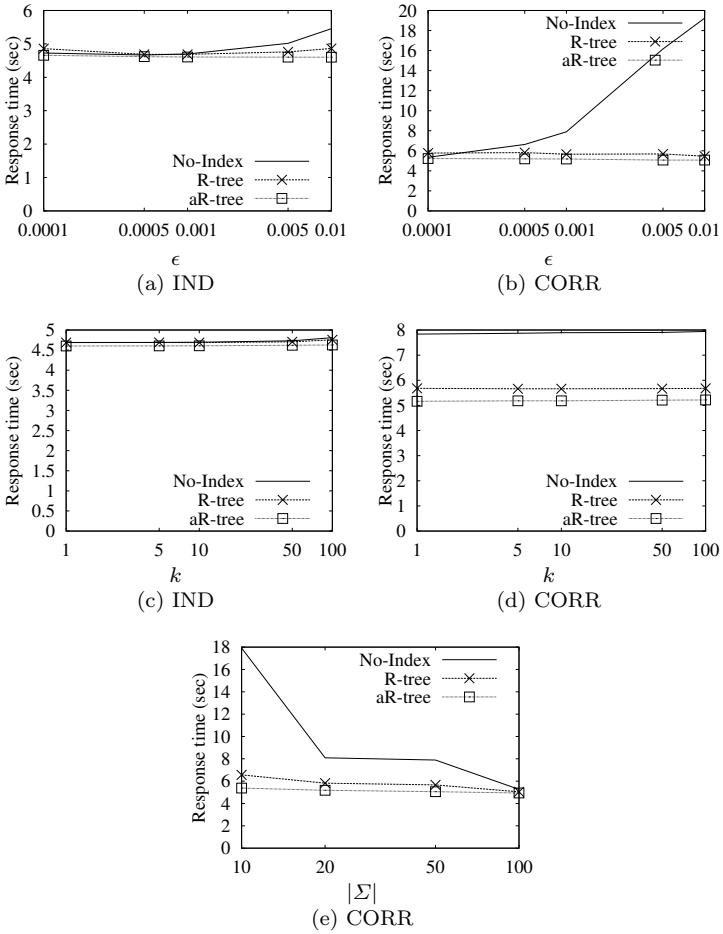


Fig. 11 Comparison of BA alternatives: scoring attributes of type IND & CORR and ISLES.

The effect of this cost is more obvious in case of CORR scoring attributes compared to IND because an overall larger number of object pairs needs to be examined. Another important observation is that the response time of the aR-tree alternative is less affected by the varying parameters, due to its ability to prune more object pairs. Finally, with the usage of more seeds for score generation, i.e., increase of $|\Sigma|$, the response time of all alternatives decreases since more object pairs have high aggregate scores.

Figure 10 reports the response time for a similar test on DFA. As expected, the aR-tree implementation always outperforms the other alternatives, in some cases for more than two orders of magnitude. This is due to the fact that the R-tree and No-Index alternatives primarily focus on the spatial predicate of k-SDJoin, and cannot fully employ the pruning power of bound θ and termina-

Table 4 Estimation of any-k depths c_R and c_S.

collection	score type	$\|R\| : \|S\|$	depth	real values	[15]	our model
HOTELS-RESTS	–	4 : 3	c_R	34,215	17,454	35,869
			c_S	28,461	17,454	26,749
ISLES	IND	1 : 1	c_R	322	262	435
			c_S	322	262	435
		2 : 1	c_R	431	283	367
			c_S	256	283	185
		3 : 1	c_R	479	304	540
			c_S	215	304	180
		4 : 1	c_R	666	308	772
			c_S	269	308	194
		5 : 1	c_R	837	327	920
			c_S	222	327	185
ISLES	CORR	1 : 1	c_R	348,295	262	300,038
			c_S	348,295	262	300,038
		2 : 1	c_R	570,376	283	551,177
			c_S	292,987	283	276,664
		3 : 1	c_R	683,732	304	671,374
			c_S	250,690	304	223,033
		4 : 1	c_R	765,469	308	792,565
			c_S	226,945	308	198,184
		5 : 1	c_R	905,738	327	886,609
			c_S	186,265	327	177,105

tion threshold T. On the other hand, due to its ability to use the score bounds and θ, T, the aR-tree based implementation not only outperforms the other two alternatives, but it is also very little affected by the increase of ϵ. In fact, the overall cost of the aR-tree implementation is dominated by the indexing time (over 95%). We also observe that all DFA alternatives perform similar on IND and CORR scoring types and are oblivious to both the number of results k and of seeds $|\Sigma|$; for the aR-tree alternative, this is because the response time is dominated by indexing cost, while for the R-tree and No-Index alternatives, this is because of primarily focusing on the spatial joining component of the queries.

Finally, we also experiment with the three alternatives for BA. The results reported in Figure 11 confirm the bounding and pruning advantages of the aR-tree evaluation compared to the R-tree and No-Index alternatives when the join involves blocks of objects.

6.4 Estimating any-k and top-k Depths

We next study the effectiveness of our model for any-k and top-k depths (Section 4.2). Tables 4 and 5 report the estimated values of c_R, c_S, d_R, d_S for default ϵ, k, $|\Sigma|$, $|R|+|S|$ while varying $|R| : |S|$. For illustration purposes, we present only the results of our tests on the ISLES collections for both IND and CORR scoring attributes; similar observations are drawn for FLICKR.

Table 5 Estimation of top-k depths d_R and d_S.

collection	score type	$\|R\| : \|S\|$	depth	real values	[15]	our model
HOTELS-RESTS	–	4 : 3	d_R	50,271	34,908	50,271
			d_S	27,886	34,908	26,114
ISLES	IND	1 : 1	d_R	5,050	523	5,653
			d_S	6,319	523	7,208
		2 : 1	d_R	7,279	566	6,482
			d_S	5,150	566	3,551
		3 : 1	d_R	8,552	608	8,773
			d_S	4,982	608	4,569
		4 : 1	d_R	13,245	617	12,485
			d_S	4,994	617	3,697
		5 : 1	d_R	17,856	654	17,813
			d_S	3,941	654	2,567
ISLES	CORR	1 : 1	d_R	465,531	523	468,544
			d_S	495,041	523	470,963
		2 : 1	d_R	849,155	566	756,636
			d_S	415,484	566	407,091
		3 : 1	d_R	909,964	608	922,063
			d_S	339,361	608	308,974
		4 : 1	d_R	1,214,600	616	1,138,258
			d_S	314,421	616	303,024
		5 : 1	d_R	1,128,750	654	1,311,250
			d_S	244,226	654	254,108

When objects are assigned IND scoring attributes, any-k depth values are solely related to the join selectivity of the input collections. Yet, the model of [15] which always sets $c_R = c_S$, manages to accurately estimate any-k depths only if $|R| = |S|$; in fact, notice how little the estimated values change while varying the $|R| : |S|$ ratio. In contrast, our model is able to cope with the increase of the cardinalities ratio and accurately estimate c_R, c_S. Overall, the average relative estimation error for [15] is 34% while for our model is 21%. On the other hand, the top-k depth values are also related to the score distribution, besides the join selectivity. As a result, the model of [15] cannot deliver good results unless the scoring attribute follows a uniform distribution and/or prior knowledge. Overall, the estimation of the [15] model has an 90% relative error over the real values, while our model only 15%.

In case of CORR scores, the model of [15] is even less accurate. The estimated c_R, c_S, d_R, d_S values are orders of magnitude off the real values On average, the relative error introduced by [15] in CORR is 100% for both c_R, c_S and d_R, d_S, while by our model is only 7% and 6%, respectively. Different from IND, with objects assigned CORR scores, a large part of the inputs is accessed for a k-SDJoin query since similar objects may have similar individual scores which also results in high aggregate scores. The model in [15] is unable to capture this behavior and, so, the estimated values of any-k and top-k depths are almost identical for both IND and CORR.

Lastly, for the HOTELS-RESTS real dataset, the model of [15] always performs worse than ours, due to the correlation property of real-world ratings

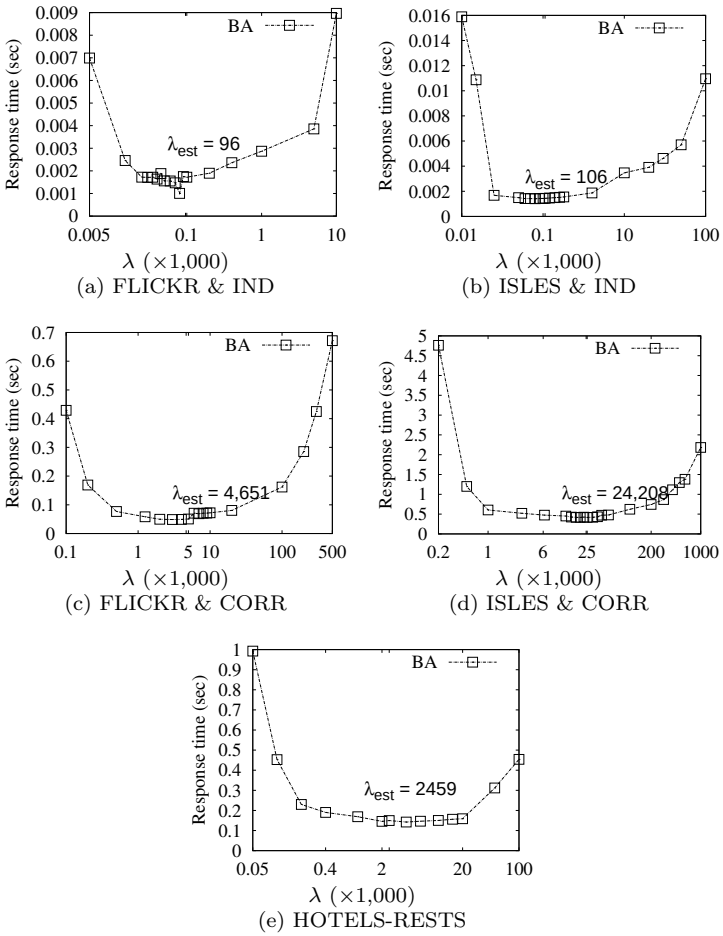
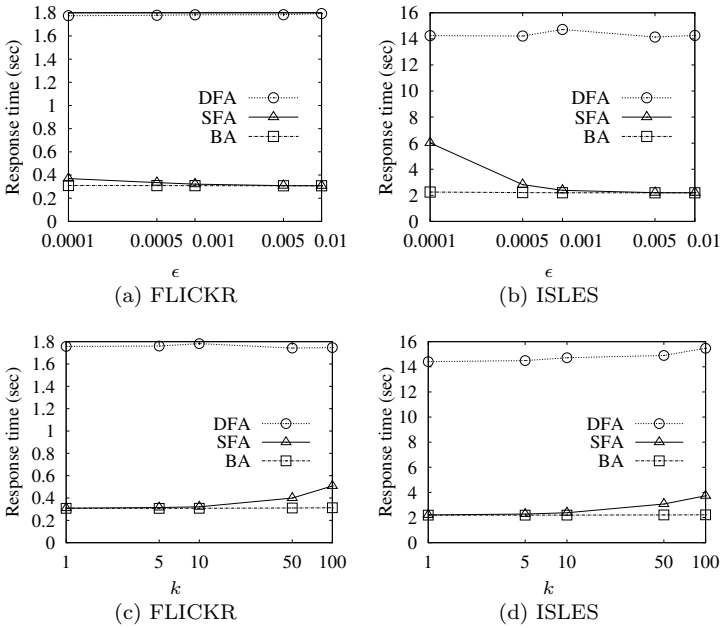


Fig. 12 Response time of BA varying λ.

as we discussed in Section 6.1. Specifically, the relative error introduced by [15] is 44% for c_R, c_S and 28% for d_R, d_S, while by our model is only 5% and 3%, respectively.

6.5 Estimating the Optimal Block Size

To investigate the accuracy of our model for estimating the optimal block size, we run BA varying the λ value inside the $[1, max\{d_r, d_S\}]$ interval. Figure 12 reports the response time of BA for both real-world and synthetic datasets; in order to clearly demonstrate the effect of tuning λ, note that the reported time in these figures does not include the cost of sorting the inputs by their scoring attribute. To make the figures readable and clear we only show the time



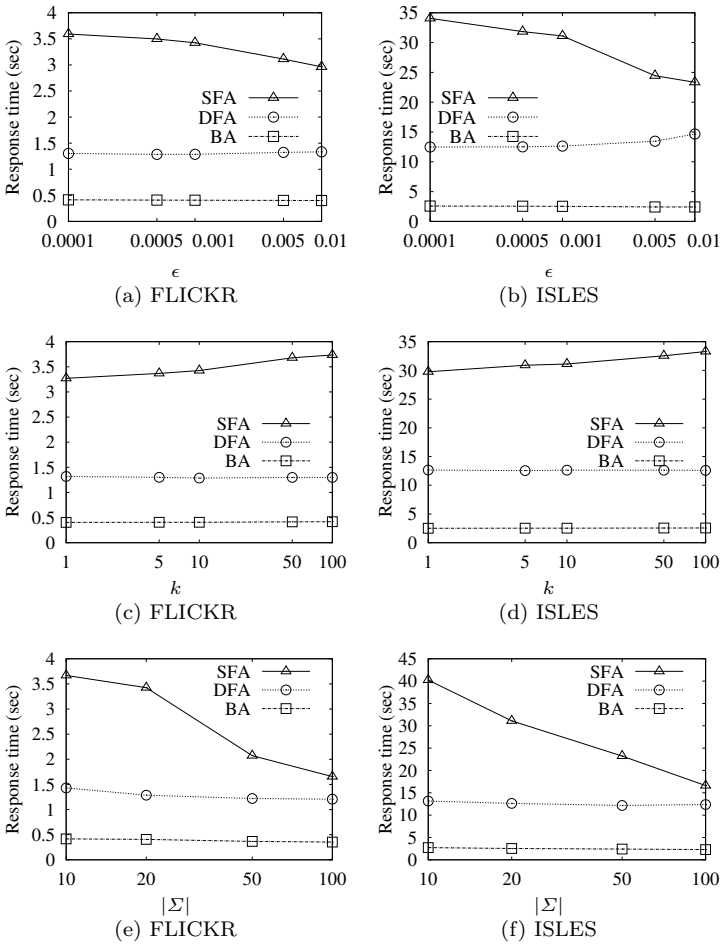
(a) FLICKR (b) ISLES

(c) FLICKR (d) ISLES

Fig. 13 Comparison of SFA, DFA and BA algorithms on scoring attributes of type IND.

around the optimal value λ_{opt} of block size and mark value λ_{est} estimated by our model. Our experiments reveal the trade-off between the response time of BA and the value of λ. Recall that for $\lambda = 1$ BA operates similar to SFA but as the block size increases towards λ_{opt} the algorithm increasingly benefits from the block-wise evaluation. However, when λ increases beyond optimal value λ_{opt}, BA becomes less efficient as it resembles an improved version of DFA which computes an increasing larger part of the spatial ϵ-distance join $R \bowtie_\epsilon S$ join. Although our model is not able to find the exact λ_{opt}, the figures show that employing λ_{est} the execution time of BA for IND and CORR score types, and for the real dataset increases only 2%, 4% and 5% on average, respectively. Note that this estimation procedure is very fast; our experiments show that the time spend to compute λ_{est} corresponds to only the 1% of the total response time of BA, on average.

Finally, we also experimented combining the model in [15] for c_R, c_S, d_R and d_S with our model for estimating optimal block size. In this case the average relative increase in the execution time of BA was 5% for IND but 19% for the real datasets and 723% for CORR, indicating that an accurate estimation of depths is critical for selecting a good value of λ.



(a) FLICKR (b) ISLES
(c) FLICKR (d) ISLES
(e) FLICKR (f) ISLES

Fig. 14 Comparison of SFA, DFA and BA algorithms on scoring attributes of type CORR.

6.6 Comparison of the Evaluation Algorithms

We next compare BA against SFA and DFA. 13 reports their response time for k-SDJoin in case of IND scoring attributes while reducing the join selectivity (i.e., increasing ϵ) and varying k. Similarly, Figure 14 reports the response times in case of CORR scores while also varying the number of seeds $|\Sigma|$. BA outperforms both SFA and DFA, in all cases. The advantage of BA is more significant for CORR scores compared to IND since larger parts of the input collections need to be accessed for computing a k-SDJoin query. Notice also that BA is more robust to the variation of the involved parameters compared to SFA and DFA. To better understand the effect of varying the join selectivity recall that k-SDJoin intuitively comes as a hybrid of a join and a top-k query, which introduces an interesting trade-off. Specifically, while increasing ϵ, the

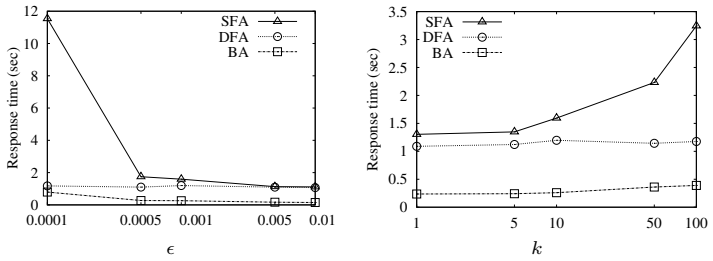


Fig. 15 Real-world HOTELS-RESTS collections.

spatial ϵ-distance join component of the query becomes less selective and thus, more expensive. However, as more object pairs qualify the join predicate, the best k results can be now identified faster, sometimes even among the highly ranked objects. In other words, the top-k component of the query becomes cheaper. In what follows, we discuss in detail how varying the query selectivity affects the response time of the evaluation methods.

Our tests are consistent with our preliminary analysis in [32]. Still, there exist two important differences. First, the block size for BA is not set by hand through repeated tests; instead, the model presented in Section 4.1 and evaluated in Section 6.5 is used. Second, we also experiment with IND scores for a more comprehensive study. We observe that BA is always the most efficient method for k-SDJoin and is very robust to the variation of the ϵ, k and $|\Sigma|$ parameters. Due to examining the objects in decreasing order of their score, both BA and SFA are positively affected by the decrease of the join selectivity (i.e., the increase of ϵ), although the benefit is more significant for SFA. It is also important to notice that SFA is faster than DFA for IND scores but slower for CORR. This is because the insert and update strategy employed by SFA to build the aR-trees over the already examined objects is in practice slower compared to the bulk loading used by DFA and BA when a large part of the collections needs to be accessed and indexed, i.e., the case of CORR scoring attributes.

When increasing parameter k, we observe that all three algorithms are negatively affected as they need to examine and compute the aggregate score for a larger number of object pairs, which qualify the distance join predicate. In other words, the algorithms compute a larger part of the $R \bowtie_\epsilon S$ join result. The effect of k is more obvious for SFA due to primarily focusing on the ranking component of k-SDJoin. In contrast, the algorithms are positively affected by the increase of $|\Sigma|$ for CORR scores. This is because the scoring and the join attribute (i.e., the spatial location) become less correlated, similar to the collections of IND scores.

We also compare BA against SFA and DFA on the real-world collections of test HOTELS-RESTS while varying the number of results k and the join selectivity (threshold ϵ). Figure 15 confirms our previous observations on the synthetic datasets; BA always outperforms SFA and DFA.

Finally, we conduct scalability and cardinality tests varying parameters $|R| + |S|$, $|R| : |S|$ for ISLES with both IND and CORR scores; Figure 16 reports the response time of the algorithms. We observe that (i) BA outperforms SFA and DFA in all cases, with the advantage being more obvious for CORR scores, and (ii) BA scales always better than DFA and in most of setups also than SFA. This is because with larger datasets (on one side for the cardinality tests or both sides for the scalability ones), it gets increasingly more expensive for SFA to update the aR-tree indexes, and for DFA to build an index for the full dataset. In those cases, the benefits of BA leveraging incremental accessing and block-based indexing and pruning become more prominent.

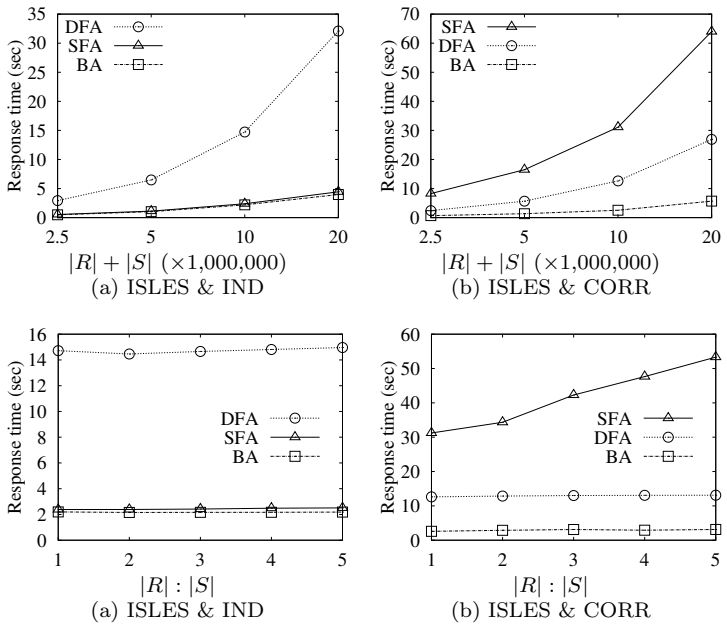


Fig. 16 Scalability and cardinality tests.

6.7 Discussion

As BA already operates in a block-wise fashion, the following three minor changes are required when the available memory is limited. First, indices only for the top blocks of the input collections are stored in main memory. These blocks contain the highest scored objects and are the most frequently accessed in practice. The remaining blocks are kept on disk and accessed only if necessary. Second, to select a block size, at least two input blocks must fit inside the available main memory. Third, the objective cost function $\mathcal{C}(\lambda)$ of Section 4.1 is extended to also consider the I/O cost for (i) reading the blocks, (ii) storing

their indices, and (iii) joining them. Finally, SFA and DFA can be also extended to work as disk-based methods, using disk-based aR-tree for k-SDJoin. However, since both algorithms use a single large aR-tree per input, additional updating and accessing costs may incur.

7 Related Work

Finally, we review previous work on ranking queries, including top-k queries, top-k joins (besides [14]), rank join depth estimation, and top-k joins on spatial data. Last, we also briefly discuss previous work on spatial joins.

7.1 Top-k Queries

Fagin et al. [8] present an analytical study of various methods for top-k aggregation of ranked inputs by monotone aggregate functions. Consider a collection 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 be also possible. For the case where both sorted and random accesses are possible, the *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 lowest seen score in source S_i; $T = \gamma(\ell_1, ..., \ell_m)$ defines a lower bound for the aggregate score of objects never seen in any S_i yet. If the k-th 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, [23] presents an optimized implementation of the *No-Random accesses Algorithm* (NRA), originally proposed also in [8]. The top-k results are incrementally fetched based on their aggregate scores. [43] presents a framework for top-k queries on top of relations having multi-attribute indices. An *index-merge* paradigm is proposed to merge multiple index nodes progressively and selectively. Recently, there has also been work that considers multiple attributes in top-k ranking criteria. For example, [33] studies the semantic based spatial keyword querying, which finds the k objects most similar to the query, subject to their spatial, textual and semantic meaning properties. To this end, the authors propose hierarchical indexing structures to integrate all types of involved information and devise appropriate pruning techniques.

7.2 Top-k Joins

Natsev et al. [25] first studied top-k join evaluation proposing multi-way join operator J*. Objects are accessed incrementally from the input streams (e.g., in round-robin) sorted by their scores. Partial join results are computed, and at each step, the top partial combination is completed by filling the missing values from the streams. J* incrementally outputs the top combinations in the heap if they are complete join results. As a follow-up to [14], Li et al. [20] applied HRJN* on multiple inputs with one or more scoring attributes each. The rank join with multiple inputs and more than one scoring attributes were also covered by PBRJ in [37]. As we discuss in Section 3.4, our BA can directly employ [10] for multiple scoring attributes, while handle multiple inputs similar to [20] or [37]. Further, [7,41,36] addressed top-k joins where the inputs originate from different physical locations. Wu et al. [41] model this as a graph problem solved by a branch-and-bound algorithm that minimizes the number of network accesses. In contrast, [7,36] determined the number of objects to be accessed from each network input, through the depth estimation procedure. Last, Ntarmos et al [27] studied top-k joins in NoSQL databases employing statistical structures (similar to 2-dimensional histograms) to reduce object accesses in a distributed environment. In this paper, we focus on the centralized scenario.

BA to previous work. In [37], the authors shortly discussed a variant of PBRJ where a block of objects is accessed at a time, instead of a single object. In practice however, this block-based variant significantly differs from our BA. Although the objects are accessed in blocks, they are still processed one-by-one as in the original HRJN* (and hence, also in SFA) and thus, current object is still probed against all buffered objects. In contrast, current block in BA is joined only with a small number of blocks from the other input. Chakrabarti et al. [4] also discussed a block-based evaluation strategy in the context of top-k keyword search where the join operator involves the intersection of compressed posting lists. The range of document ids is split into intervals and an upper aggregate score bound is defined for each interval. List intersection is performed at the interval level using score bounds to enable interval-based pruning. Compared to BA, the work in [4] differs in two ways: (i) it primarily focuses on minimizing the decompression cost for the posting lists and (ii) the interval-based partitioning and join are strongly related to the problem at hand, i.e., keyword search, and cannot be applied to other type of join attributes and predicates such as k-SDJoin.

7.3 Rank Join Depth Estimation

Our work is also related to studies [7,15,20,38] on estimating the *depth* of rank join operators, i.e., the number of objects accessed from each input. Compared to our own study in Section 4.2, these works either make specific assumptions about the objects or employ expensive statistics. Ilyas et al. [15] proposed

a probability-based model assuming that the join and scoring attributes are uniformly distributed and independent from each other. Li et al. [20] proposed a sampling-based approach where the termination score is estimated by a rank join on uniformly sampled data. Such a method usually overestimates the depth, especially if both the number of results k and the sampling ratio are small. Schnaitter et al. [38] addressed the above issues assuming that the distributions of the scoring and the join attribute are known in advance, and then used to determine the number of join results with a specific aggregate score. To this end, the authors employ multi-dimensional histograms [31]. Similar statistics (2-dimensional histograms) are used in [7]. However, such approaches cannot be adopted for k-SDJoin as multi-dimensional histograms are efficient to compute and accurate only for join attributes of small domains and simple join predicates such as equality. Note that, in the absence of such statistics, [38] assumes uniform and independent distributions similar to [15].

7.4 Spatial Top-k Joins

Previous work on top-k joins for spatial data significantly differs from our k-SDJoin operator. Specifically, the "top-k spatial join" in [46] retrieves k objects in collection R intersecting the maximum number of objects in S. Thus, ranking is based on spatial intersections and not on the aggregation of (non-spatial) scores as in k-SDJoin. Similarly, "top-k similarity join" in [18, 42] and "top-k spatio-textual similarity join" in [13] differ from a k-SDJoin, as ranking is based on the similarity of join attributes. Last, "proximity rank join" over objects that carry a feature vector and a scoring attribute in [24], also differs from k-SDJoin as (i) it additionally involves a query object and (ii) the ranking criterion combines the scoring attributes and the distance of the objects to each other and to the query object.

To our knowledge, the only work closely related to k-SDJoin is [21], where each object o (e.g., a biological cell) is assigned a set of probabilistic locations and a confidence p_o for belonging to a specific cell class. The score of an (r, s) pair is defined by their confidence probabilities p_r, p_s and the distance between their uncertain locations. In contrast, the aggregate score function for k-SDJoin does not involve the distance of the objects, but the distance is used as the join predicate. Further, the solution proposed in [21] is of limited applicability as it is bound to a specific aggregation function and can efficiently work only with the L_1 distance.

7.5 Spatial Joins

There has been ample research on spatial joins [16], due to their wide applicability and their high complexity compared to simpler operations such as selections or nearest neighbor search. Most works focus on the evaluation of *spatial intersection* joins, where the objective is to find pairs of objects from

two input collections which share at least one common point in space. Early research focused on developing algorithms for disk-based data which are either raw [1,29] or indexed. More recently, the focus shifted to distributed and parallel algorithms for spatial joins [34,44], as well as methods that take advantage of the large memories in contemporary commodity hardware [26].

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 [3]. 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 [5]. 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 [12,39] the results are reported one-by-one in ascending order of their spatial distance. For non-incremental computation of closest pairs, [6] extends the nearest neighbor algorithm of [35] achieving in this way, minimum memory requirements and better access locality for tree nodes.

8 Conclusions

In this paper, we introduced the top-k spatial distance join (k-SDJoin). For its efficient evaluation, we first presented the Score-First (SFA) and Distance-First (DFA) algorithms based on existing literature, which prioritize either the score ranking or the spatial joining component of a k-SDJoin query, respectively. Then, we proposed a novel evaluation method termed the Block-based Algorithm (BA), which alleviates the weaknesses of SFA and DFA by operating in a block-wise fashion. We also devised a model to automatically select an appropriate block size for BA, while as a side contribution, we proposed a novel depth estimation model which considers arbitrary score distributions and correlations between the score and the spatial location of the objects. We optimized all three algorithms by employing aR-trees to allow both spatial distance and score-based pruning. Our extensive experimental analysis on real and synthetic data confirmed (i) the effectiveness of our automatic block size tuning procedure, and (ii) the peformance superiority of BA over both SFA and DFA.

In the future, we plan to investigate the efficient evaluation of k-SDJoin in distributed environments and in the case of multiple inputs. As a different direction, we also intend to study the application of the top-k join operator

on other types of join attributes and predicates, e.g., overlap joins in temporal databases or textual similarity joins.

References

1. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Scalable sweeping-based spatial join. In: VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA, pp. 570–581 (1998)
2. Belussi, A., Faloutsos, C.: Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In: VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland., pp. 299–310 (1995)
3. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using R-trees. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993., pp. 237–246 (1993)
4. Chakrabarti, K., Chaudhuri, S., Ganti, V.: Interval-based pruning for top-k processing over compressed lists. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany, pp. 709–720 (2011)
5. Chan, E.P.F.: Buffer queries. IEEE TKDE **15**(4), 895–910 (2003)
6. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA., pp. 189–200 (2000)
7. Doulkeridis, C., Vlachou, A., Kotidis, Y., Polyzotis, N.: Processing of rank joins in highly distributed systems. In: IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012, pp. 606–617 (2012)
8. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA, pp. 102–113 (2001)
9. Faloutsos, C., Seeger, B., Traina, A., Traina Jr., C.: Spatial join selectivity using power laws. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA., pp. 177–188 (2000)
10. Finger, J., Polyzotis, N.: Robust and efficient algorithms for rank join evaluation. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009, pp. 415–428 (2009)
11. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984, pp. 47–57 (1984)
12. Hjaltason, G.R., Samet, H.: Incremental distance join algorithms for spatial databases. In: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA., pp. 237–248 (1998)
13. Hu, H., Li, G., Bao, Z., Feng, J., Wu, Y., Gong, Z., Xu, Y.: Top-k spatio-textual similarity join. IEEE TKDE **28**(2), 551–565 (2016)
14. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. In: VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany, pp. 754–765 (2003)
15. Ilyas, I.F., Shah, R., Aref, W.G., Vitter, J.S., Elmagarmid, A.K.: Rank-aware query optimization. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004, pp. 203–214 (2004)
16. Jacox, E.H., Samet, H.: Spatial join techniques. ACM Transactions on Database Systems **32**(1), 7 (2007)
17. Kiefer, J.: Sequential minimax search for a maximum. Proceedings of the American Mathematical Society **4**(3), 502–506 (1953)

18. Kim, Y., Shim, K.: Parallel top-k similarity join algorithms using mapreduce. In: IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012, pp. 510–521 (2012)
19. Koudas, N., Muthukrishnan, S., Srivastava, D.: Optimal histograms for hierarchical range queries. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA, pp. 196–204 (2000)
20. Li, C., Chang, K.C.C., Ilyas, I.F., Song, S.: Ranksql: Query algebra and optimization for relational top-k queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005, pp. 131–142 (2005)
21. Ljosa, V., Singh, A.K.: Top-k spatial joins of probabilistic objects. In: Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico, pp. 566–575 (2008)
22. Mamoulis, N., Papadias, D.: Multiway spatial joins. ACM Transactions on Database Systems **26**(4), 424–475 (2001)
23. Mamoulis, N., Yiu, M.L., Cheng, K.H., Cheung, D.W.: Efficient top-k aggregation of ranked inputs. ACM TODS **32**(3) (2007)
24. Martinenghi, D., Tagliasacchi, M.: Proximity rank join. PVLDB **3**(1), 352–363 (2010)
25. Natsev, A., Chang, Y.C., Smith, J.R., Li, C.S., Vitter, J.S.: Supporting incremental join queries on ranked inputs. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, pp. 281–290 (2001)
26. Nobari, S., Tauheed, F., Heinis, T., Karras, P., Bressan, S., Ailamaki, A.: TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013, pp. 701–712 (2013)
27. Ntarmos, N., Patlakas, I., Triantafillou, P.: Rank join queries in nosql databases. PVLDB **7**(7), 493–504 (2014)
28. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP operations in spatial data warehouses. In: Advances in Spatial and Temporal Databases, 7th International Symposium, SSTD 2001, Redondo Beach, CA, USA, July 12-15, 2001, Proceedings, pp. 443–459 (2001)
29. Patel, J.M., DeWitt, D.J.: Partition based spatial-merge join. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996., pp. 259–270 (1996)
30. Petersen, S.B., Neves-Petersen, M.T., Henriksen, S.B., Mortensen, R.J., Geertz-Hansen, H.M.: Scale-free behaviour of amino acid pair interactions in folded proteins. PLoS ONE **7**(7) (2012)
31. Poosala, V., Haas, P.J., Ioannidis, Y.E., Shekita, E.J.: Improved histograms for selectivity estimation of range predicates. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996., pp. 294–305 (1996)
32. Qi, S., Bouros, P., Mamoulis, N.: Efficient top-k spatial distance joins. In: Advances in Spatial and Temporal Databases - 13th International Symposium, SSTD 2013, Munich, Germany, August 21-23, 2013. Proceedings, pp. 1–18 (2013)
33. Qian, Z., Xu, J., Zheng, K., Zhao, P., Zhou, X.: Semantic-aware top-k spatial keyword queries. World Wide Web **21**(3), 573–594 (2018)
34. Ray, S., Simion, B., Brown, A.D., Johnson, R.: Skew-resistant parallel in-memory spatial join. In: Conference on Scientific and Statistical Database Management, SSDBM'14, Aalborg, Denmark, June 30 - July 02, 2014, pp. 6:1–6:12 (2014)
35. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995., pp. 71–79 (1995)
36. Saouk, M., Doulkeridis, C., Vlachou, A., Nørvåg, K.: Efficient processing of top-k joins in mapreduce. In: 2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016, pp. 570–577 (2016)
37. Schnaitter, K., Polyzotis, N.: Optimal algorithms for evaluating rank joins in database systems. ACM TODS **35**(1) (2010)

38. Schnaitter, K., Spiegel, J., Polyzotis, N.: Depth estimation for ranking query optimization. In: Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007, pp. 902–913 (2007)
39. Shin, H., Moon, B., Lee, S.: Adaptive multi-stage distance join processing. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA., pp. 343–354 (2000)
40. Smith, A.J.: Sequentiality and prefetching in database systems. ACM TODS **3**(3), 223–247 (1978)
41. Wu, M., Berti-Équille, L., Marian, A., Procopiuc, C.M., Srivastava, D.: Processing top-k join queries. PVLDB **3**(1-2), 860–870 (2010)
42. Xiao, C., Wang, W., Lin, X., Shang, H.: Top-k set similarity joins. In: Proceedings of the 2009 IEEE International Conference on Data Engineering, pp. 916–927 (2009)
43. Xin, D., Han, J., Chang, K.C.: Progressive and selective merge: computing top-k with ad-hoc ranking functions. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007, pp. 103–114 (2007)
44. Zhang, S., Han, J., Liu, Z., Wang, K., Xu, Z.: SJMR: parallelizing spatial join with mapreduce on clusters. In: Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31 - September 4, 2009, New Orleans, Louisiana, USA, pp. 1–8 (2009)
45. Zhao, K., Zhou, S., Tan, K.L., Zhou, A.: Supporting ranked join in peer-to-peer networks. In: 16th International Workshop on Database and Expert Systems Applications (DEXA'05), pp. 796–800 (2005)
46. Zhu, M., Papadias, D., Lee, D.L., Zhang, J.: Top-k spatial joins. IEEE TKDE **17**(4), 567–579 (2005)