Title	Feat-SKSJ: Fast and Exact Algorithm for Top-k Spatial-Keyword Similarity Join
Author(s)	Amagata, Daichi; Tsuruoka, Shohei; Arai, Yusuke et al.
Citation	GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems. 2021, p. 15-24
Version Type	AM
URL	https://hdl.handle.net/11094/92846
rights	© 2021 ACM.This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Amagata D., Tsuruoka S., Arai Y., et al. Feat-SKSJ: Fast and Exact Algorithm for Top-k Spatial-Keyword Similarity Join. GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, 15 (2021); https://doi.org/10.1145/3474717.3483629.
Note	

Feat-SKSJ: Fast and Exact Algorithm for Top-k Spatial-Keyword Similarity Join

Daichi Amagata*
Osaka University, PRESTO
Japan
amagata.daichi@ist.osaka-u.ac.jp

Shohei Tsuruoka*
Osaka University
Japan
tsuruoka.shohei@ist.osaka-u.ac.jp

Yusuke Arai
Osaka University
Japan
arai.yusuke@ist.osaka-u.ac.jp

Takahiro Hara
Osaka University
Japan
hara@ist.osaka-u.ac.jp

ABSTRACT

Due to the proliferation of GPS-enabled mobile devices and IoT environments, location-based services are generating a large number of objects that contain both spatial and keyword information, and spatial-keyword databases are receiving much attention. This paper addresses the problem of top-k spatial-keyword similarity join, which outputs k object pairs with the highest similarity. This query is a primitive operator for important applications, including duplicate detection, recommendation, and clustering.

The main bottleneck of the top-k spatial-keyword similarity join is to compute the similarity of a given object pair. To avoid this computation as much as possible, a state-of-the-art algorithm utilizes a filter that can skip the exact similarity computation of a given pair. However, this algorithm suffers from a loose threshold at the first stage, a high filtering cost, and the impossibility of filtering many pairs in a batch. We propose Feat-SKSJ, which removes these drawbacks and quickly outputs the exact result. Extensive experiments on real datasets show that Feat-SKSJ is significantly faster than the state-of-the-art algorithm.

CCS CONCEPTS

• **Information systems → Proximity search**.

KEYWORDS

spatial-keyword data, similarity join

ACM Reference Format:
Daichi Amagata, Shohei Tsuruoka, Yusuke Arai, and Takahiro Hara. 2018. Feat-SKSJ: Fast and Exact Algorithm for Top-k Spatial-Keyword Similarity Join. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

*Both authors contributed equally to this research.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Due to the proliferation of IoT technologies and GPS-enabled mobile devices (i.e., smartphones), location-based services are becoming more ubiquitous. Real-world examples include Google Maps, OpenStreetMap, and Geo-tagging in social networking services (e.g., Facebook and Instagram). The users/applications of these services are generating a large number of spatial-keyword objects that contain both geo-spatial information (e.g., check-in locations) and keyword (textual) information (e.g., hashtags) [12]. Because these objects can be used to improve service quality, gain benefits, and detect events by offering spatial-keyword similarity search operations [8, 14, 16, 20, 29, 33–35, 43, 50, 51], spatial-keyword databases are receiving much attention. However, real-world spatial-keyword databases usually face the following observation: Objects in the databases are usually collected from multiple (different) sources [3, 21, 22], thus similar or essentially the same contents are posted by different users [31]. Some objects are hence duplicated or represented by similar but different information (because of GPS errors and typing errors). Because of this observation, spatial-keyword databases contain many redundant objects, which would degrade the quality of spatial-keyword similarity search results. It is therefore important to clean up the databases by removing the redundancy. A spatial-keyword similarity join [7], which outputs pairs of similar objects w.r.t. spatial and keyword similarity, achieves this.

In [7, 21, 22, 28], the threshold-based spatial-keyword similarity join was considered. Given a threshold for spatial similarity and a threshold for keyword similarity, two objects are similar iff their spatial and keyword similarities are not less than the thresholds. The threshold-based spatial-keyword similarity join outputs all such pairs. However, specifying the two thresholds is not an easy task for general users (and even for experts), because these thresholds are domain specific and cannot control the result size. If the threshold(s) is (are) low, the result size may become huge, which overwhelms the users. On the other hand, if the threshold(s) is (are) high, users may have a few pairs as a result, and this does not help applications. To avoid this issue, the top-k spatial-keyword similarity join was considered in [18]. This operation does not require the two thresholds as input, and it outputs the k most similar pairs, so that users can obtain their required result size. In this paper, we address the top-k spatial-keyword similarity join problem.

State-of-the-art. The main bottleneck of the top-k spatial-keyword similarity join is the exact computation of similarity of a given object pair. To avoid the exact similarity computation as much as possible, [18] proposed a signature-based filtering algorithm, SigJoin. Informally, for each object $o \in O$, where O is a set of objects, this algorithm generates its signature set based on an intermediate threshold of the top-k join result. This is a set of pairs of a keyword held by o and a spatial region containing the coordinate of o. For two objects $o, o' \in O$, if their signature sets have no intersection, it is easy to see that they are not similar, thus $\langle o, o' \rangle$ cannot be the top-k result. Based on this idea, SigJoin skips the similarity computation between objects with no signature set intersection.

However, SigJoin has drawbacks that degrade its efficiency. First, to obtain the first threshold for the top-k join result, SigJoin uses random k object pairs (in a node of a spatial index). Because this threshold is loose, the pruning efficiency of signature sets degrades and SigJoin computes the similarities of many unnecessary object pairs. Second, the cost of computing signature sets is not small, and they cannot be obtained in a pre-processing phase because they are dependent on an intermediate threshold. Last, SigJoin cannot prune object pairs in $O_i \times O_j$, where $O_i, O_j \subset O$, in a batch.

Our contribution. To quickly compute the exact top-k spatial-keyword similarity join result without suffering from the above issues, we propose Feat-SKSJ (fast and exact algorithm for top-k spatial-keyword similarity join). This algorithm does not employ an online indexing approach like signature sets generation, and it supports fast top-k spatial-keyword similarity join processing through the following techniques derived from a data structure based on an aggregate R-tree (aR-tree) [27] built offline: (i) it computes a tight threshold with a small cost in the first stage, (ii) given $O_i, O_j \subset O$, it filters all pairs $\in O_i \times O_j$ in a batch with $O(1)$ time (if they are guaranteed not to be top-k), and (iii) given o and O_i, it filters all pairs $\langle o, o' \rangle$, where $o' \in O_i$, in a batch, with no signature set generation (if they cannot be top-k).

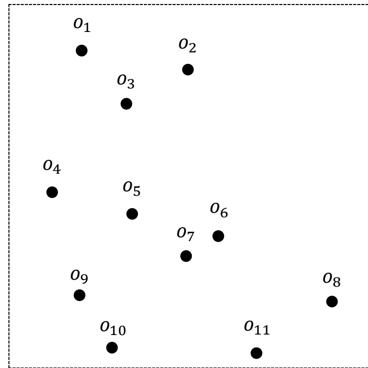
To summarize, this paper makes the following contributions:

- We propose a simple yet effective data structure by extending aR-tree to enable batch filtering of unnecessary object pairs. We also introduce an efficient algorithm that builds this data structure.
- We propose Feat-SKSJ, which employs the above data structure for exact top-k spatial-keyword similarity join.
- We conduct experiments on real datasets, and the experimental results show that Feat-SKSJ significantly outperforms the state-of-the-art algorithm SigJoin.

Organization. The rest of this paper is organized as follows. Section 2 formally defines our problem. Section 3 presents Feat-SKSJ. We report our experimental results in Section 4, and Section 5 reviews related works. Last, Section 6 concludes this paper.

2 PRELIMINARY

Let O be a static set of spatial-keyword objects. A spatial-keyword object $o \in O$ is represented as $o = \langle p, s \rangle$, where $o.p$ is the two-dimensional coordinate of o and $o.s$ is the set of keywords held by o. Hereafter, object and spatial-keyword object are used interchangeably. To measure the similarity between two objects o_i and o_j, we need to consider spatial similarity and keyword set similarity. We first define spatial similarity.



Object o	Keywords $o.s$
o_1	w_1, w_3
o_2	w_2, w_5
o_3	w_1, w_2, w_4
o_4	w_3, w_7, w_8
o_5	w_2, w_6, w_7, w_9
o_6	w_1, w_2, w_5, w_7
o_7	w_1, w_2, w_6, w_7, w_9
o_8	w_1, w_8
o_9	w_3
o_{10}	w_1, w_3
o_{11}	w_2, w_5

Figure 1: Example of a set of spatial-keyword objects. Given $k = 1$ and $\alpha = 0.5$, the top-1 spatial-keyword similarity join returns $\langle o_5, o_7 \rangle$ as the result.

Definition 1 (Spatial similarity). *Given objects o_i and o_j, their spatial similarity, $sim_p(o_i, o_j)$, is defined as:*

$$sim_p(o_i, o_j) = 1 - \frac{dist(o_i.p, o_j.p)}{dist_{max}},$$

where $dist(o_i.p, o_j.p)$ is the Euclidean distance between $o_i.p$ and $o_j.p$ and $dist_{max}$ is the maximum distance in the space of O.

Notice that we have $sim_p(o_i, o_j) \in [0, 1]$. Next, we consider keyword set similarity. As the standard measure of set similarity is Jaccard similarity [18, 23], we also use it to measure the keyword set similarity[1].

Definition 2 (Keyword set similarity). *Given objects o_i and o_j, their keyword set similarity, $sim_s(o_i, o_j)$, is defined as:*

$$sim_s(o_i, o_j) = \frac{|o_i.s \cap o_j.s|}{|o_i.s \cup o_j.s|}.$$

Then, the spatial-keyword similarity between two objects is:

Definition 3 (Spatial-keyword similarity). *Given objects o_i and o_j and a weighting parameter $\alpha \in [0, 1]$, their spatial-keyword similarity, $sim(o_i, o_j)$, is*

$$sim(o_i, o_j) = \alpha \cdot sim_p(o_i, o_j) + (1 - \alpha) sim_s(o_i, o_j). \quad (1)$$

A large (small) α weights the spatial (keyword set) similarity more. How to set this parameter is application-dependent. Now we are ready to define our problem.

Definition 4 (Top-k spatial-keyword similarity join). *Given a set of objects O, a weighting parameter α, and a result size k, the top-k spatial-keyword similarity join outputs k pairs of different objects (i.e., $\langle o_i, o_j \rangle$ s.t. $i \neq j$) with the highest similarity computed by Equation (1) among $O \times O$ (ties are broken arbitrarily).*

Example 1. *Figure 1 illustrates an example of O and the table at right in this figure shows the set of keywords w held by each object in O.*

[1] Our solution can support Cosine and Dice similarities. In Cosine and Dice cases, $sim_s = \frac{|o_i.s \cap o_j.s|}{\sqrt{|o_i.s| \cdot |o_j.s|}}$ and $sim_s = \frac{2 \cdot |o_i.s \cap o_j.s|}{|o_i.s| + |o_j.s|}$, respectively.

Table 1: Notations frequently used in this paper

Notation	Meaning
O	Set of spatial-keyword objects
o	Spatial-keyword object
$sim_p(\cdot,\cdot)$	Spatial similarity between objects
$sim_s(\cdot,\cdot)$	Keyword set similarity between objects
$sim(\cdot,\cdot)$	Similarity between objects
k	Join result size
α	Weighting factor $\in [0,1]$
τ	Intermediate threshold of the top-k result
n_i	Node of an akR-tree
R_i	Minimum bonding rectangle of n_i
J_i	$\max sim_s(\cdot,\cdot)$ among object pairs in sub-tree of n_i
S_i	Set of keywords held by objects maintained in n_i

Consider top-1 spatial-keyword similarity join on O where $\alpha = 0.5$ and $dist_{max} = 18$. Assume that $dist(o_5, o_7) = 3$ and $dist(o_6, o_7) = 1.8$. Then, $sim_p(o_5, o_7) = 1-3/18 = 0.83$ and $sim_p(o_6, o_7) = 1-1.8/18 = 0.9$. On one hand, $sim_s(o_5, o_7) = 4/5 = 0.8$ and $sim_s(o_6, o_7) = 3/6 = 0.5$. Hence, $sim(o_5, o_7) = 0.5 \cdot 0.83 + 0.5 \cdot 0.8 = 0.815$ and $sim(o_6, o_7) = 0.5 \cdot 0.9 + 0.5 * 0.5 = 0.7$. The join result is $\langle o_5, o_7 \rangle$.

As with [18], we assume that O is memory-resident. The objective of this paper is to devise a fast algorithm that returns the exact answer of the top-k spatial-keyword similarity join. Table 1 summarizes the notations frequently used in this paper.

3 FEAT-SKSJ

To quickly process a top-k spatial-keyword similarity join, we should avoid unnecessary similarity computations. SigJoin [18] utilizes a signature set to achieve this. However, its filtering cost is still high. Besides, its filter is conducted only between an object o and a subset of O, that is, the signature set filtering cannot prune pairs in $O_i \times O_j$, where $O_i, O_j \subset O$, at one time.

Main idea. To achieve fast top-k spatial-keyword similarity join, it is better to prune many object pairs, which cannot be in the top-k result, in a batch *with a small computational cost.* More specifically, a filtering technique that can prune all pairs in $O_i \times O_j$ in a batch is desirable (if they cannot be top-k pairs). Given a threshold of the top-k join result τ, this filtering can be achieved if we can estimate an upper-bound similarity of all object pairs in $O_i \times O_j$. We now face several challenges. A tight threshold is required to enable filtering of the object pairs in $O_i \times O_j$ (a loose threshold would fail to prune the object pairs). However, how to compute such a tight threshold *with a small computational cost* is not trivial (obtaining a tight threshold with an expensive cost is meaningless). In addition, how to efficiently obtain an upper-bound similarity of all object pairs in $O_i \times O_j$ is also not trivial.

To overcome these non-trivial challenges, we first extend aggregate R-tree (aR-tree) [27], so that this data structure can efficiently support both tight threshold computation and upper-bound similarity computation. For tight threshold computation, we propose a model to estimate aR-tree nodes that contain object pairs with high similarity. For upper-bound computation, we utilize two observations: (i) The R-tree structure can compute a lower-bound distance between two nodes, which derives an upper-bound spatial similarity. (ii) An upper-bound Jaccard similarity of object pairs in $O_i \times O_j$ can be computed *offline*. Hence, by maintaining this Jaccard similarity as the aggregate value of a node in the aR-tree, we can compute an upper-bound similarity of all object pairs in $O_i \times O_j$ with $O(1)$ time[2]. We design the extended aR-tree to filter the pairs in $o \times O_i$. Then, we propose Feat-SKSJ, which exploits the extended aR-tree to efficiently obtain the exact top-k join result.

Overview. We build the extended aR-tree, called akR-tree (aggregation value and keywords R-tree) in a pre-processing (offline) phase[3]. Note that this phase is done only once, and the akR-tree supports any k and α.

Given k and α, Feat-SKSJ computes the top-k join result by using the following techniques:

(1) Threshold-Initialization(O, k, α): Feat-SKSJ first computes a tight threshold τ by identifying the akR-tree nodes that would have object pairs with high similarities.
(2) Node-Node-Filtering(O_i, O_j, α, τ): Given $O_i, O_j \subset O$, Feat-SKSJ filters all object pairs in $O_i \times O_j$ iff their upper-bound similarities are not larger than τ.
(3) Object-Node-Filtering(o, O_i, α, τ): Given o and O_i, Feat-SKSJ filters all object pairs in $o \times O_i$ iff their upper-bound similarities are not larger than τ.

In Section 3.1, we describe the detailed structure of akR-tree[4]. Section 3.2 introduces Node-Node-Filtering$(\cdot,\cdot,k,\alpha,\tau)$ and Object-Node-Filtering$(\cdot,\cdot,k,\alpha,\tau)$. We present our offline algorithm in Section 3.3. Section 3.4 details our online algorithm Feat-SKSJ along with Threshold-Initialization(O, k, α).

3.1 Data Structure

We elaborate the structure of akR-tree. It is essentially an aR-tree, and the main difference between aR-tree and akR-tree is that the leaf nodes of the akR-tree store a set of keywords appearing in the objects maintained in them. Given a set O_i of the objects maintained by the sub-tree rooted at node n_i of the akR-tree, n_i maintains the following components.

- R_i: the minimum bounding rectangle (MBR) that encloses the objects in O_i.
- J_i: $\max_{o,o' \in O_i} sim_s(o, o')$.
- S_i: a set of the keywords appearing in O_i (this set is held only by a leaf node).

Note that J_i and S_i are utilized for Node-Node-Filtering$(\cdot,\cdot,\alpha,\tau)$ and Object-Node-Filtering$(\cdot,\cdot,\alpha,\tau)$, respectively.

[2]Notice that the distance computation is done in $O(1)$ time.
[3]As with existing works, e.g., [18, 26, 32], this index is also memory-resident. We confirmed that the memory size of the akR-tree on a dataset of a million objects is less than 200 MBytes, which easily fits into main-memory.
[4]Although SigJoin also uses a spatial-index, how to exploit a spatial index is totally different from ours. The objective of using a spatial index in SigJoin is to generate a signature set. On the other hand, we utilize an akR-tree to enable the efficient computation of a tight threshold and batch filtering of object pairs in different nodes, which are not supported by SigJoin. In addition, SigJoin needs to compute a signature set for each pair of an object and a node. Our solution can compute an upper-bound similarity between an object and a leaf node without incurring an additional indexing cost, unlike signature set generation.

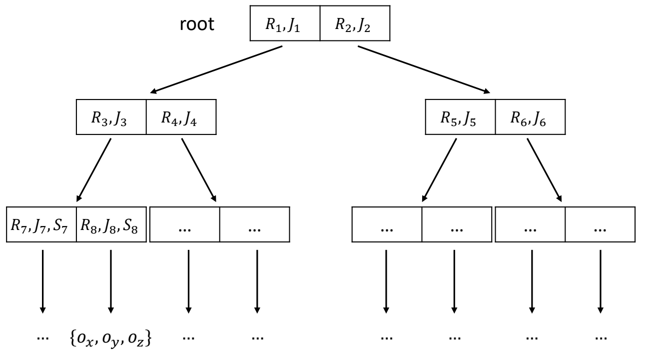
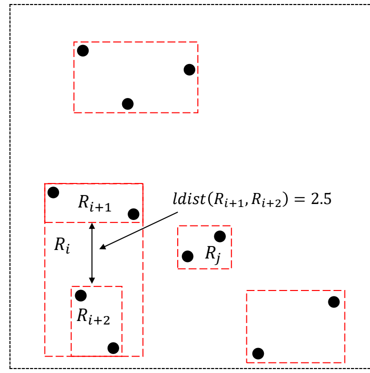


Figure 2: Example of an akR-tree. R_i represents an MBR, J_i represents an aggregate value (Jaccard similarity), and S_i represents a set of keywords appearing in O_i.



MBR	J
R_i	0.5
R_{i+1}	0.17
R_{i+2}	0.5

Figure 3: Example of NODE-NODE-FILTERING$(O_{i+1}, O_{i+2}, \alpha, \tau)$, where $\alpha = 0.5$, $\tau = 0.7$, and $ldist(R_{i+1}, R_{i+2} = 2.5)$. O is from Example 1.

EXAMPLE 2. *Figure 2 illustrates a brief example of an akR-tree. Each intermediate node n_i maintains $\langle R_i, J_i \rangle$, and each leaf node $n_{i'}$ maintains $\langle R_{i'}, J_{i'}, S_{i'} \rangle$. For example, R_8 is the MBR of $\{o_x, o_y, o_z\}$, $J_8 = \max\{sim_s(o_x, o_y), sim_s(o_x, o_z), sim_s(o_y, o_z)\}$, and $S_8 = o_x.s \cup o_y.s \cup o_z.s$. Nodes n_1 to n_6 are intermediate nodes, thus they do not have S_i.*

3.2 Our Filtering

Next, we present how to achieve NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$ and OBJECT-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$. In this section, we assume that a threshold τ is given (how to do this is introduced in Section 3.4). Furthermore, we use O_i to denote the set of objects maintained in the sub-tree of the akR-tree rooted at node n_i. We below demonstrate that, many object pairs can be efficiently filtered in a batch by exploiting the akR-tree structure, which has not been devised so far.

NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$. Given two leaf nodes n_i and n_j, we first introduce a filtering technique that can prune all object pairs in $O_i \times O_j$ at one time. Its idea is simple: we compute their upper-bound similarity, as stated earlier. To derive this upper-bound similarity, we define some notations below. Let $ldist(R_i, R_j)$ be the minimum distance between two MBRs R_i and R_j. Given n_i and n_j, they have common ancestors, and let $n_{i,j}$ be their ancestor with the maximum depth. Moreover, let

$$usim(O_i, O_j) = \alpha(1 - \frac{ldist(R_i, R_j)}{dist_{max}}) + (1 - \alpha)J_{i,j}. \quad (2)$$

Then, we have:

LEMMA 1. *Consider a threshold τ and two leaf nodes of an akR-tree, n_i and n_j. If $usim(O_i, O_j) < \tau$, all object pairs in $O_i \times O_j$ cannot be the top-k join result.*

PROOF. Given an arbitrary object pair $\langle o, o' \rangle$ where $o \in O_i$ and $o' \in O_j$, it is trivial that $dist(o.p, o'.p) \geq ldist(R_i, R_j)$. From the definition, we have $J_{i,j} \geq \max_{o_a, o_b \in O_i \cup O_j} sim_s(o_a, o_b)$, thus $J_{i,j} \geq sim_s(o, o')$. It is now clear that $sim(o, o') \leq usim(O_i, O_j)$. Therefore, if $usim(O_i, O_j) < \tau$, we have $sim(o, o') < \tau$, guaranteeing that this lemma holds. □

EXAMPLE 3. *An example of NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$ is illustrated in Figure 3. The set of objects is from Example 1. The rectangles with dashed lines represent MBRs, R_i, R_{i+1}, and R_{i+2}, where R_{i+1} and R_{i+2} are children of R_i. The table at right shows J_i. For instance, $J_{i+2} = sim_s(o_4, o_5) = 1/6 = 0.17$ and $J_{i+2} = sim_s(o_9, o_{10}) = 1/2 = 0.5$, so $J_i = 0.5$.*

Assume $\alpha = 0.5$ and $\tau = 0.7$. The minimum distance between R_{i+1} and R_{i+2}, $ldist(R_{i+1}, R_{i+2})$, is 0.25. Hence, $usim(R_{i+1}, R_{i+2}) = 0.5 \cdot (1 - 2.5/18) + 0.5 \cdot 0.5 = 0.68$. Because $usim(R_{i+1}, R_{i+2}) < \tau$, all object pairs in $O_i \times O_j$ are pruned.

Recall that Equation (2) proves that one operation of NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$ needs only $O(1)$ time.

OBJECT-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$. Consider a pair of leaf nodes $\langle n_i, n_j \rangle$ that are not filtered by NODE-NODE-FILTERING(O_i, O_j, α, τ). Given an object $o \in O_i$, it is possible that all object pairs in $o \times O_j$ cannot be the top-k join result. We below present how to identify this observation.

The idea is again to upper-bound the similarities of $o \times O_j$. We use S_j and the minimum distance between o and R_j to derive an upper-bound similarity. Let $ldist(o, R_j)$ be the minimum distance between o and R_j. Furthermore, let

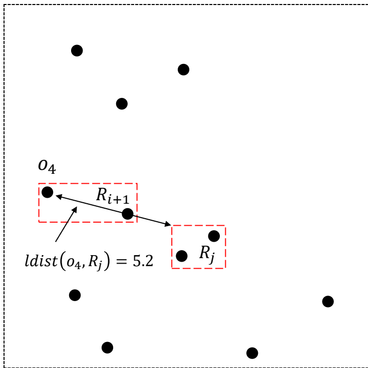
$$usim(o, O_j) = \alpha(1 - \frac{ldist(o, R_j)}{dist_{max}}) + (1 - \alpha)\frac{|o.s \cap S_j|}{\min\{|o.s|, |S_j|\}}. \quad (3)$$

We have:

LEMMA 2. *Given an object $o \in O_i$ and a set of objects $O_j \subset O$, all object pairs in $o \times O_j$ cannot be the top-k join result if $usim(o, R_j) < \tau$.*

PROOF. Trivially, we have $ldist(o, R_j) \leq dist(o, o')$ for all $o' \in O_j$. Recall that $S_j = \bigcup_{o' \in O_j} o.s$. We hence have $|o.s \cap S_j| \geq |o.s \cap o'.s|$ for all $o' \in O_j$. Besides, $\min\{|o.s|, |S_j|\} \leq |o.s \cup o'.s|$ for all $o' \in O_j$, meaning that $\frac{|o.s \cap S_j|}{\min\{|o.s|, |S_j|\}} \geq sim_s(o, o')$ for all $o' \in O_j$. It is now clear that $usim(o, O_j) \geq sim(o, o')$ for an arbitrary object $o' \in O_j$. Therefore, this lemma holds. □

EXAMPLE 4. *An example of OBJECT-NODE-FILTERING(o_5, O_j, α, τ) is illustrated in Figure 4. The setting is similar to that of Example 3, and the table at right shows S_j. We again assume $\alpha = 0.5$ and $\tau = 0.7$.*



MBR	S
R_j	$w_1, w_2, w_5, w_6, w_7, w_9$

Figure 4: Example of OBJECT-NODE-FILTERING(o_4, O_j, α, τ), where $\alpha = 0.5$, $\tau = 0.7$, and $ldist(o_4, R_j) = 5.2$. O is from Example 1.

The minimum distance between o_4 and R_j, $ldist(o_4, R_j)$, is 5.2. We have $\min\{|o_4.s|, |S_j|\} = |o_4.s| = 3$. Hence, $usim(o_4, O_j) = 0.5 \cdot (1 - 5.2/18) + 0.5 \cdot 1/3 = 0.52$. Because $usim(o_4, O_j) < \tau$, all object pairs in $o_4 \times O_j$ are pruned.

Compared with NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$, OBJECT-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$ is a bit costly, since it needs to compute a set intersection, as shown in Equation (3). However, it is still efficient and effective, because it can prune all object pairs in $o \times O_j$ (if possible) with a one-time set intersection.

3.3 Pre-processing

We have demonstrated the effectiveness of akR-tree through Lemmas 1 and 2, which can efficiently prune many object pairs in a batch with $O(1)$ time or a one-time set intersection computation. This section introduces how to build an akR-tree efficiently, as how to efficiently obtain J_i for each node is not trivial. Our main idea here is to exploit a state-of-the-art set similarity join algorithm [45].

Algorithm 1: PRE-PROCESSING

Input: O
Output: An akR-tree
Build an R-tree
for *each leaf node* n_a **do**
    Compute S_a by scanning $\bigcup_{o \in O_a} o.s$
    Compute J_a via PP-Join$(\bigcup_{o \in O_a} o.s, 0)$ [45]
for *each intermediate node* n_b *(bottom-up order)* **do**
    Compute J_b via PP-Join$(\bigcup_{o \in O_b} o.s, \max_{n_i \in n_b.C} J_i)$ [45]

Algorithm 1 describes the pre-processing algorithm that builds an akR-tree. We first build an R-tree. For each leaf node n_a of the R-tree, we compute $S_a = \bigcup_{o \in O_a} o.s$ and $J_a = \max_{o, o' \in O_a} sim_s(o.s, o'.s)$. To efficiently compute this, we run the state-of-the-art set similarity join algorithm PP-Join [45] on O_a[5]. Next, for each intermediate node n_b, we compute J_b in a bottom-up manner. Let $n_b.C$ be a set of children of n_b, and consider $n_i \in n_b.C$. Because $O_i \subseteq O_b$, we have $J_i \leq J_b$. From this observation, by setting $\max_{n_i \in n_b.C} J_i$ as a threshold, J_b can be efficiently obtained by PP-Join[6]. (At intermediate nodes, we do not compute keyword set similarity between objects that have been already compared.) In this way, each node n_b obtains J_b. After that, we obtain the akR-tree.

3.4 Join Processing

We have clarified how to efficiently obtain the akR-tree. The remaining challenge is to efficiently obtain a tight threshold for effective filtering.

THRESHOLD-INITIALIZATION(O, k, α). We again exploit the akR-tree to obtain a tight τ. Let p_i^l and p_i^u be the lower-left and the upper-right points of the MBR R_i, respectively. Define $f(n_i)$ as

$$f(n_i) = \alpha \cdot (1 - \frac{dist(p_i^l, p_i^u)}{dist_{max}}) + (1 - \alpha) J_i. \quad (4)$$

From this equation, it can be seen that, if R_i is small, $f(n_i)$ becomes high. Similarly, if J_i is high, $f(n_i)$ becomes high. That is, leaf nodes with these features potentially contain object pairs with high similarity.

Based on this model (observation), we obtain a tight threshold from these leaf nodes. Algorithm 2 details our threshold initialization algorithm. Let N be a set of all leaf nodes of the akR-tree. We compute $f(n_i)$ for all $n_i \in N$, and maintain l leaf nodes in N_l that have the highest $f(\cdot)$. Then, we conduct a self-join on O_i with the l-highest rank of $f(\cdot)$, to initialize τ and T (a set of top-k object pairs found so far). We set l empirically and $l = O(k)$ in our implementation.

Algorithm 2: THRESHOLD-INITIALIZATION

Input: O, k, α, and an akR-tree of O
$N \leftarrow$ a set of leaf nodes
$N_l \leftarrow \emptyset$
for *each leaf node* $n_i \in N$ **do**
    Compute $f(n_i)$ though Equation (4)
    Update N_l to have l nodes with the highest $f(\cdot)$ so far
$\tau \leftarrow 0$, $T \leftarrow \emptyset$
for *each* $n_j \in N_l$ **do**
    for *each* $o \in O_j$ **do**
        for *each* $o' \in O_j$ *where* $o' \neq o$ **do**
            Compute $sim(o, o')$
            Update T and τ
Return $\langle \tau, T \rangle$

EXAMPLE 5. *We describe an example of Algorithm 2 by using Figure 3 ($\alpha = 0.5$ and $k = 1$). It first computes $f(\cdot)$ for each leaf node (red rectangles). Assume $l = 1$. Obviously, R_j has the largest $f(\cdot)$, so it runs a self-join on O_j. As a result, we have $sim(o_6, o_7) = 0.7$, rendering $\tau = 0.7$ and $T = \langle o_6, o_7 \rangle$.*

[5] At a leaf node n_i, we do not have a threshold of J_i. However, $|O_i|$ is small, so J_i can be quickly obtained.

[6] If $J_i = 1$ at an arbitrary node, we do not run PP-Join on its ancestors.

Algorithm 3: FEAT-SKSJ

```
   Input: O, k, α, and an akR-tree of O
   Output: T (k object pairs with the highest similarity)
1  ⟨τ, T⟩ ← THRESHOLD-INITIALIZATION(O, k, α)
2  for each leaf node n_i of the akR-tree (i ∈ [1, |N| − 1]) do
3      for each leaf node n_j of the akR-tree (j ∈ [i, |N|]) do
4          f ← NODE-NODE-FILTERING(O_i, O_j, α, τ)
5          if f = 0 then
6              for each o ∈ O_i do
7                  f ← OBJECT-NODE-FILTERING(o, O_j, α, τ)
8                  if f = 0 then
9                      for each o′ ∈ O_j do
10                         if sim(o, o′) > τ then
11                             Update T and τ
```

It is important to recall that, as introduced in Section 1, applications of spatial-keyword similarity join have many near-duplicated objects, which tend to fall into the same leaf nodes of the akR-tree. Therefore, the "local" joins can find a tight threshold by enumerating a much smaller number of object pairs than $\frac{|O|(|O|-1)}{2}$. That is, our threshold initialization cost, which is $O(|N| + l|O_j|^2)$ time[7], is much cheaper than the overall cost, as shown in Section 4.4.

Feat-SKSJ. Now we are ready to introduce our online algorithm, and Algorithm 3 summarizes Feat-SKSJ. Given k and α, Feat-SKSJ initializes T and τ through THRESHOLD-INITIALIZATION(O, k, α). Then, given a pair of leaf nodes $\langle n_i, n_j \rangle$, Feat-SKSJ tests NODE-NODE-FILTERING(O_i, O_j, α, τ). All object pairs in $O_i \times O_j$ are ignored if $usim(O_i, O_j) \leq \tau$. Otherwise, for each $o \in O_i$, Feat-SKSJ tests OBJECT-NODE-FILTERING(o, O_j, α, τ). Feat-SKSJ ignores all object pairs in $o \times O_j$ if $usim(o, O_j) \leq \tau$. Otherwise, Feat-SKSJ computes $sim(o, o')$ for each $o' \in O_j$ while updating T and τ. The above operations are repeated for each pair of leaf nodes.

Note that, if $n_i \in N_l$ (i.e., n_i is used for obtaining the first threshold in THRESHOLD-INITIALIZATION(O, k, α)), we set $f = 1$ at line 4 for $\langle n_i, n_i \rangle$. From this and Lemmas 1 and 2, the correctness of Feat-SKSJ is obvious.

Analysis. Recall that N is a set of leaf nodes of the akR-tree. THRESHOLD-INITIALIZATION(O, k, α) needs $O(|N| + l|O_j|^2)$ time. Feat-SKSJ tests NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$ for each pair of leaf nodes, which requires $O(|N|^2)$ time. Let P be a set of leaf node pairs that are not pruned by NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$. Note that $|P| = (1 - \epsilon_1)|N|^2$, where ϵ_1 is the pruning rate of NODE-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$. The total time of OBJECT-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$ is $O(\sum_P c_u |O_i|)$, where c_u is the average cost of computing Equation (3). Last, the total time of similarity computation (except for that in THRESHOLD-INITIALIZATION(O, k, α)) is $O(\sum_P c_{sim}(1 - \epsilon_2)|O_i||O_j|)$, where c_{sim} is the average cost of computing Equation (1) and ϵ_2 is the pruning rate of OBJECT-NODE-FILTERING$(\cdot, \cdot, \alpha, \tau)$.

[7] Given a fixed node capacity of the akR-tree, $|O_j| = O(1)$, because O_j is held by a leaf node. Therefore, the initialization time becomes $O(|N|)$.

4 EXPERIMENT

This section reports our experimental results. All experiments were conducted on a Ubuntu 16.04 LTS machine with 3.00GHz Intel Xeon Gold 6154 CPU and 512GB RAM.

4.1 Setting

Datasets. We used two real datasets.

- Places[8]: A set of public places inside the United States. Each object consists of its geo-location and a set of keywords.
- Twitter[9]: A set of geo-tagged tweets located inside the United States.

The cardinality of these datasets is 1,000,000. The number of distinct keywords in Places (Twitter) is 26,407 (277,849), and the average number of keywords held by an object in Places (Twitter) is 2.9 (4.4).

Algorithms. We evaluated

- SigJoin [18]: the state-of-the-art algorithm for the top-k spatial-keyword similarity join, and
- Feat-SKSJ: our solution proposed in this paper.

These algorithms run on a single thread. Both the algorithms were implemented in C++ and complied by g++ 5.4.0 with -O3 flag.

Literature [18] confirmed that the other existing techniques, which can deal with our problem (through some extensions), are clearly outperformed by SigJoin. We therefore do not consider them.

Parameters. Table 2 shows our parameter setting, and the bold values show the default ones. When investigating the impact of a given parameter, the other parameters were fixed at the default values. As in Definition 2, we used Jaccard similarity to measure keyword set similarity by default, but the Cosine and Dice similarity cases are also investigated in Section 4.8.

Table 2: Configuration of parameters

Parameter	Values
Cardinality of dataset [$\times 10^6$]	0.25, 0.5, 0.75, **1.0**
k	10, 50, **100**, 500, 1000
α	0.2, 0.3, 0.4, **0.5**, 0.6, 0.7, 0.8

4.2 Pre-processing Time

We first clarify that the time for building our data structure akR-tree is reasonable. On Places and Twitter, the pre-processing times were respectively 136.60 and 140.30 seconds, i.e., the akR-tree can be built within a few minutes. Recall that the pre-processing is done only once (i.e., the akR-tree is general to any k and α), thus building the akR-tree is not a bottleneck.

4.3 Tuning l

Hereafter, we report the online time(s) of Feat-SKSJ (and SigJoin). We tune l in Algorithm 2 by using $k = 50$, $k = 100$, and $k = 1000$. We used $l = k/8$, $k/4$, $k/2$, and k. Table 3 shows the running time

[8] https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces
[9] http://www.ntu.edu.sg/home/gaocong/datacode.html

Table 3: Running time of Feat-SKSJ with tuning l by using $k = 50$, $k = 100$, and $k = 1000$

	$k = 50$		$k = 100$		$k = 1000$	
l	Places	Twitter	Places	Twitter	Places	Twitter
$k/8$	323.52	292.09	183.57	586.26	4107.91	943.97
$k/4$	229.00	301.97	204.19	577.87	3602.85	376.66
$k/2$	268.20	282.55	96.56	150.28	3965.53	333.93
k	85.30	140.32	98.21	173.78	3939.41	649.42

of Feat-SKSJ with different l. When l is small (e.g., $l = k/8$), the threshold is not tight enough, so the running time is not minimized. When l is large (i.e., $l = k$) and k is large, the running time tends to be longer, particularly on Twitter. This is because Threshold-Initialization incurs many local joins. On the other hand, when k is small, l should be as large as k, to obtain a tight threshold. From this result, we set $l = \lceil k/2 \rceil$ ($l = k$) when $k \geq 100$ ($k < 100$).

4.4 Effectiveness and Efficiency of Our Approaches

Effectiveness. To demonstrate that Threshold-Initialization contributes to fast top-k join processing, we compare Feat-SKSJ with its variant, denoted by Feat-SKSJ-rand, which computes the first threshold from random k object pairs. Table 4 shows the comparison result. It can be seen that Feat-SKSJ is clearly faster than Feat-SKSJ-rand. For example, on Places, Feat-SKSJ achieves about 5.8 times speed-up against Feat-SKSJ-rand. When we obtain a threshold from random k object pairs, the threshold becomes loose, so the filtering power becomes lower. This incurs a larger number of Object-Node-Filtering and similarity computations between objects. Feat-SKSJ avoids this issue by obtaining a tight threshold at first, thus it is faster than Feat-SKSJ-rand.

We next compare Feat-SKSJ-rand with SigJoin to evaluate the effectiveness of our filtering. The experimental result demonstrates that our filtering works much better than the signature-based filtering of SigJoin, because the running time of Feat-SKSJ-rand is much faster than that of SigJoin. This performance difference is mainly derived from our Node-Node-Filtering (which is not implemented in SigJoin). This filter can prune all object pairs in $O_i \times O_j$ in $O(1)$ time, significantly reducing the numbers of Object-Node-Filtering and similarity computations.

Efficiency. To show the efficiency of each component of Feat-SKSJ, we studied the decomposed time of Feat-SKSJ. Table 5 shows the time of Threshold-Initialization (Algorithm 2), total time of Node-Node-Filtering, total time of Object-Node-Filtering, and total time of similarity computations (except for those in Threshold-Initialization), denoted by Similarity-Computation.

The first observation is that Threshold-Initialization provides a tight threshold with a small cost, since it does not dominate the running time while yielding shorter time than the other algorithms (see Table 4), on both Places and Twitter. Next, the time difference of Threshold-Initialization between Places and Twitter is derived from the difference in the number of objects in the l leaf nodes. We also observe that Node-Node-Filtering is one of the main costs of the running time, whereas Object-Node-Filtering

Table 4: Running time [sec] of Feat-SKSJ, Feat-SKSJ-rand, and SigJoin

Datasets	Feat-SKSJ	Feat-SKSJ-rand	SigJoin
Places	96.56	564.43	21671.26
Twitter	150.28	243.40	14136.43

Table 5: Decomposed time [sec] of Feat-SKSJ

Algorithm	Places	Twitter
Threshold-Initialization	1.29	19.56
Node-Node-Filtering	76.50	111.92
Object-Node-Filtering	4.21	14.23
Similarity-Computation	14.56	4.57

incurs a smaller cost. Recall that Node-Node-Filtering incurs $O(|N|^2)$ time, but the time for Object-Node-Filtering is dependent on the pruning rate of Node-Node-Filtering. This suggests that the pruning rate is high.

4.5 Impact of Cardinality

Next, we study the scalability of Feat-SKSJ by varying the cardinality via random sampling. Figure 5 depicts the running time of Feat-SKSJ and SigJoin with different cardinality. From this figure, we see that Feat-SKSJ is much more scalable than SigJoin on both datasets. For example, Feat-SKSJ is about 224 (94) times faster than SigJoin on Places (Twitter) when the cardinality is 1 million. As demonstrated in Tables 4 and 5, the performance difference is derived from Threshold-Initialization, which provides a tight threshold with a low computational cost, and Node-Node-Filtering, which prunes many unnecessary object pairs in a batch. Thanks to these observations, Feat-SKSJ can output the top-k join result with a much less number of similarity computations than SigJoin.

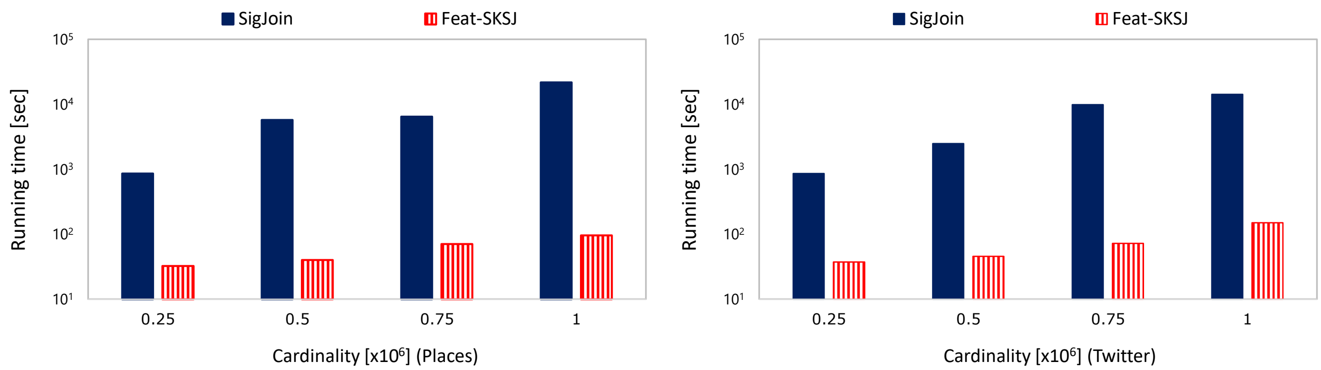
4.6 Impact of Result Size

We investigated the influence of k, and Figure 6 shows the experimental result. The running time of SigJoin is almost not affected by k. Its threshold is too loose until many similarity computations are done, so its pruning efficiency is not high even when k is small on both datasets. On the other hand, the running time of Feat-SKSJ increases as k increases. Because the threshold obtained in Threshold-Initialization becomes small in the case of large k, the search space also becomes larger in that case.

We notice that, when k is large, the running time of Feat-SKSJ on Places is very different from that on Twitter. This is also observed in Table 3. We found that Places has many object pairs with similarities that are close to the k-th highest one, compared with Twitter. Because of this nature, it is hard to prune them, so even Feat-SKSJ needs to compute their similarities, and this increases its running time.

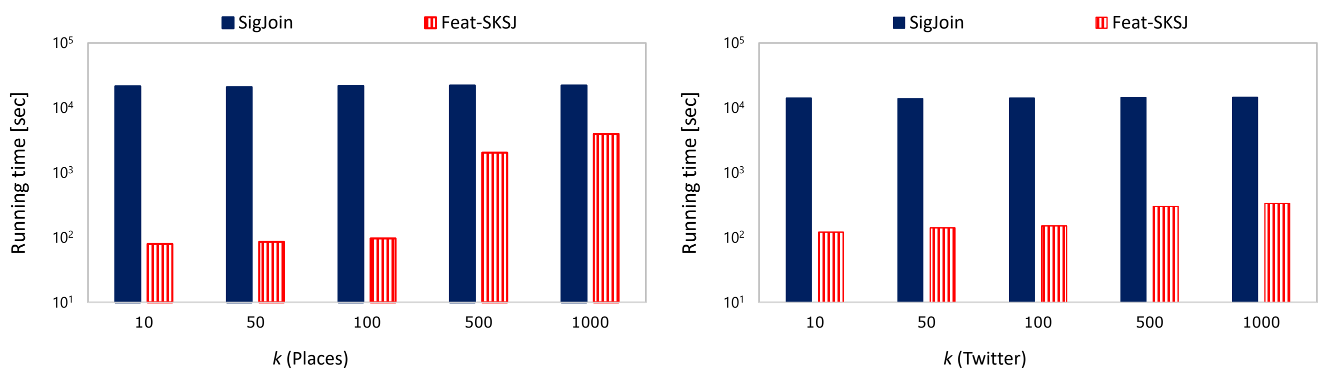
4.7 Impact of Weighting Factor

We next studied the impact of α, and Figure 7 shows the result. We see that SigJoin has a similar observation to that in Figure 6. For



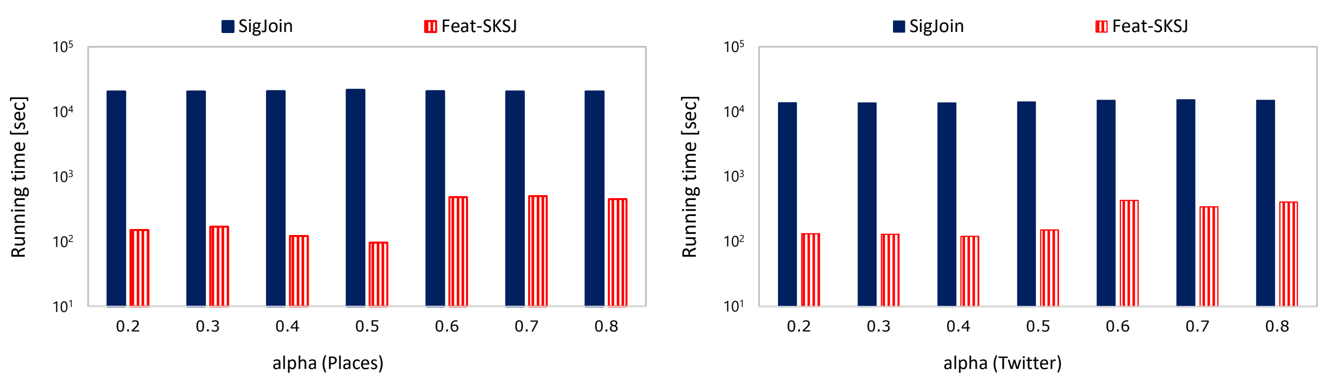
(a) Places (b) Twitter

Figure 5: Impact of cardinality of dataset



(a) Places (b) Twitter

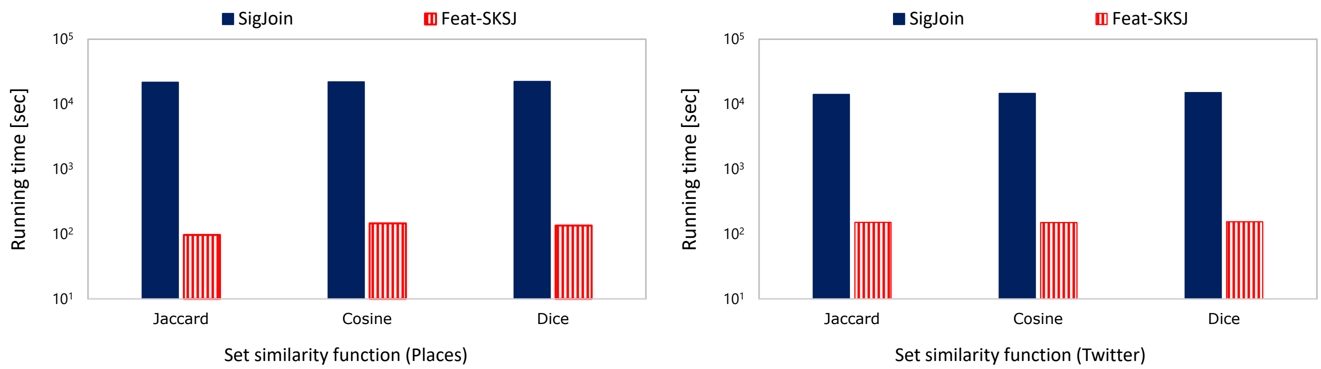
Figure 6: Impact of k (result size)



(a) Places (b) Twitter

Figure 7: Impact of α (weighting factor)

Feat-SKSJ, we can see that its running time becomes shorter when α is small. Recall that a small α weights the keyword set similarity. When α is small, object pairs with close distance and similar keyword sets tend to have high similarities. The number of such object pairs is much smaller than $\frac{|O|(|O|-1)}{2}$ (i.e., all object pairs), but Threshold-Initialization can obtain them. This renders high



(a) Places

(b) Twitter

Figure 8: Impact of set similarity function

filtering efficiency, resulting in a short running time. On the other hand, a large α weights the spatial similarity. The number of object pairs with close distance is generally large, so the filtering efficiency tends to degrade.

4.8 Impact of Set Similarity Function

Last, we compare the performance difference between keyword set similarities, i.e., Jaccard, Cosine, and Dice similarity cases. Figure 8 illustrates the comparison result. We see that the SigJoin and Feat-SKSJ show similar performances between the keyword set similarities. Because two sets with high Jaccard similarity also have high Cosine or Dice similarity, the pruning efficiencies of SigJoin and Feat-SKSJ are not affected by the set similarity functions. Actually, this result is consistent with those reported in existing works, e.g., [25, 39].

5 RELATED WORK

Spatial-keyword join is a primitive operator not only for duplicate detection but also for location-based recommendation [28] and clustering [6]. Section 1 has already introduced the state-of-the-art [18], so we review works on spatial-keyword join with different settings from ours.

Literatures [7, 21, 22] addressed the problem of threshold-based spatial-keyword similarity join. The first work that addressed this problem [7] extended the set similarity join-based algorithm [46] to fit spatial-keyword data. [7] used a grid for a spatial index so that the set similarity join algorithm is invoked on a small set of objects. Note that [18] empirically demonstrated that the threshold-based spatial-keyword similarity join technique is outperformed by SigJoin. In [21, 22], signature-based filtering was proposed. These papers actually assume a spatial *region* (not point) as spatial information and optimized their technique for this assumption. Therefore, their technique is hard to be employed in our problem. Some works addressed the threshold-based spatial-keyword similarity join on MapReduce environments [5, 28]. Their techniques focus on data partitioning, which is irrelevant to our setting (i.e., a single thread).

Spatial-keyword search on static data. Spatial-keyword search has been extensively studied so far, and most works addressed the problem of top-k spatial-keyword similarity search [13, 14, 29, 34, 42, 43, 50]. These works designed indexes that integrate spatial and keyword indexes, and an experimental paper [10] compared their performances. Although our solution also utilizes an index that integrates spatial and keyword information, our approach is different from the search techniques. Recall that our solution exploits the index to filter many pairs of objects that cannot be top-k in a batch, which is not considered in the similarity search problem. The search techniques can support our problem by iteratively conducting a top-k spatial-keyword similarity search for each object. However, this approach is quite expensive, as demonstrated in [18].

Spatial-keyword monitoring on dynamic data. Some applications, such as Pub/Sub systems, consider spatial-keyword data streams [1, 9, 19, 24, 41]. Given a set of continuous spatial-keyword queries and a dynamic set of objects, they monitor the k most similar objects for each query. They optimize techniques that can filter unnecessary *queries* for a new object. Some works assume a distributed system [11, 36, 40], and [25] assumes mobile queries. These techniques may help to update the result of a top-k spatial-keyword similarity join on dynamic data. This setting is left for future work.

Spatial join. Given a dataset and a radius threshold r, a spatial join computes all pairs of spatial points with distances that are not larger than r [2, 26, 30, 32]. Existing approaches to spatial join are essentially nested-loop on a spatial index. Some works [15, 17] addressed the closest pair queries in spatial databases. These techniques do not consider keyword set similarity, so we cannot employ them for our problem.

Set similarity join has been widely studied in a single thread setting [38, 39, 44, 46, 48], dynamic sets [4], and distributed setting [37, 47]. To efficiently process a join query, several filtering techniques, such as length filter, prefix filter, and suffix filter, were proposed. Because these techniques do not consider spatial information, they cannot be used for our problem directly. Literature [23] conducted extensive experiments to study the performances of these filters and shows that PP-Join [45], which employs length and prefix filters, works the best. Therefore, Feat-SKSJ utilizes PP-Join to support efficient building of its index.

6 CONCLUSION

Motivated by the fact that spatial-keyword databases are becoming increasingly important for many practical applications, we addressed the problem of top-k spatial-keyword similarity join, an important operator in spatial-keyword databases. This paper proposed a new solution for this problem, Feat-SKSJ, which employs a data structure based on an aggregate R-tree. From this data structure, Feat-SKSJ can obtain a tight threshold with a small cost, filter all object pairs between two different nodes in a batch with $O(1)$ time, and filter all pairs of a given object and objects in a node with a one-time computation of set intersection. We conducted experiments on two real datasets, and the experimental results show that Feat-SKSJ significantly outperforms the state-of-the-art algorithm SigJoin.

This work devised a fast algorithm on a single thread setting. To further accelerate query processing efficiency, optimizations for parallelization approaches based on multicore and distributed computing environments, such as Spark [49], can be considered. It is worth addressing such optimizations in a future work.

REFERENCES

[1] Abdulaziz Almaslukh and Amr Magdy. 2018. Evaluating spatial-keyword queries on streaming data. In *SIGSPATIAL*. 209–218.
[2] Daichi Amagata and Takahiro Hara. 2019. Identifying the Most Interactive Object in Spatial Databases. In *ICDE*. 1286–1297.
[3] Daichi Amagata, Takahiro Hara, and Shojiro Nishio. 2015. Distributed top-k query processing on multi-dimensional data with keywords. In *SSDBM*. 10:1–10:12.
[4] Daichi Amagata, Takahiro Hara, and Chuan Xiao. 2019. Dynamic Set kNN Self-Join. In *ICDE*. 818–829.
[5] Jaime Ballesteros, Ariel Cary, and Naphtali Rishe. 2011. Spsjoin: parallel spatial similarity joins. In *SIGSPATIAL*. 481–484.
[6] Alexandros Belesiotis, Dimitrios Skoutas, Christodoulos Efstathiades, Vassilis Kaffes, and Dieter Pfoser. 2018. Spatio-textual user matching and clustering based on set similarity joins. *The VLDB Journal* 27, 3 (2018), 297–320.
[7] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. 2012. Spatio-textual similarity joins. *PVLDB* 6, 1 (2012), 1–12.
[8] Xin Cao, Gao Cong, Christian S Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *SIGMOD*. 373–384.
[9] Lisi Chen, Gao Cong, Xin Cao, and Kian-Lee Tan. 2015. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*. 255–266.
[10] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. 2013. Spatial keyword query processing: an experimental evaluation. *PVLDB* 6, 3 (2013), 217–228.
[11] Yue Chen, Zhida Chen, Gao Cong, Ahmed R Mahmood, and Walid G Aref. 2020. SSTD: a distributed system on streaming spatio-textual data. *PVLDB* 13, 12 (2020), 2284–2296.
[12] Zhida Chen, Lisi Chen, Gao Cong, and Christian S Jensen. 2021. Location-and keyword-based querying of geo-textual data: a survey. *The VLDB Journal* (2021), 1–38.
[13] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. 2011. Text vs. space: efficient geo-search query processing. In *CIKM*. 423–432.
[14] Gao Cong, Christian S Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB* 2, 1 (2009), 337–348.
[15] Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis, and Michael Vassilakopoulos. 2000. Closest pair queries in spatial databases. *ACM SIGMOD Record* 29, 2 (2000), 189–200.
[16] Christos Doulkeridis, Akrivi Vlachou, Dimitris Mpestas, and Nikos Mamoulis. 2017. Parallel and Distributed Processing of Spatial Preference Queries using Keywords.. In *EDBT*. 318–329.
[17] Gísli R Hjaltason and Hanan Samet. 1998. Incremental distance join algorithms for spatial databases. In *SIGMOD*. 237–248.
[18] Huiqi Hu, Guoliang Li, Zhifeng Bao, Jianhua Feng, Yongwei Wu, Zhiguo Gong, and Yaoqiang Xu. 2016. Top-k spatio-textual similarity join. *IEEE Transactions on Knowledge and Data Engineering* 28, 2 (2016), 551–565.
[19] Jiafeng Hu, Reynold Cheng, Dingming Wu, and Beihong Jin. 2015. Efficient top-k subscription matching for location-aware publish/subscribe. In *SSTD*. 333–351.
[20] Junling Liu, Ke Deng, Huanliang Sun, Yu Ge, Xiaofang Zhou, and Christian S Jensen. 2017. Clue-based spatio-textual query. *PVLDB* 10, 5 (2017), 529–540.
[21] Sitong Liu, Guoliang Li, and Jianhua Feng. 2012. Star-join: spatio-textual similarity join. In *CIKM*. 2194–2198.
[22] Sitong Liu, Guoliang Li, and Jianhua Feng. 2014. A prefix-filter based method for spatio-textual similarity join. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2354–2367.
[23] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. 2016. An empirical evaluation of set similarity join techniques. *PVLDB* 9, 9 (2016), 636–647.
[24] Shunya Nishio, Daichi Amagata, and Takahiro Hara. 2017. Geo-Social Keyword Top-k Data Monitoring over Sliding Window. In *DEXA*. 409–424.
[25] Shunya Nishio, Daichi Amagata, and Takahiro Hara. 2020. Lamps: Location-aware moving top-K pub/sub. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[26] Sadegh Nobari, Farhan Tauheed, Thomas Heinis, Panagiotis Karras, Stéphane Bressan, and Anastasia Ailamaki. 2013. TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In *SIGMOD*. 701–712.
[27] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. 2001. Efficient OLAP operations in spatial data warehouses. In *SSTD*. 443–459.
[28] Jinfeng Rao, Jimmy Lin, and Hanan Samet. 2014. Partitioning strategies for spatio-textual similarity join. In *SIGSPATIAL Workshop*. 40–49.
[29] João B Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. 2011. Efficient processing of top-k spatial keyword queries. In *SSTD*. 205–222.
[30] Darius Šidlauskas and Christian S Jensen. 2014. Spatial joins in main memory: Implementation matters! *PVLDB* 8, 1 (2014), 97–100.
[31] Anders Skovsgaard, Darius Sidlauskas, and Christian S Jensen. 2014. Scalable top-k spatio-temporal term querying. In *ICDE*. 148–159.
[32] Benjamin Sowell, Marcos Vaz Salles, Tuan Cao, Alan Demers, and Johannes Gehrke. 2013. An experimental analysis of iterated spatial joins in main memory. *PVLDB* 6, 14 (2013), 1882–1893.
[33] Naoya Taguchi, Daichi Amagata, and Takahiro Hara. 2017. Geo-social keyword Skyline queries. In *DEXA*. 425–435.
[34] Yufei Tao and Cheng Sheng. 2013. Fast nearest neighbor search with keywords. *IEEE transactions on knowledge and data engineering* 26, 4 (2013), 878–888.
[35] George Tsatsanifos and Akrivi Vlachou. 2015. On Processing Top-k Spatio-Textual Preference Queries.. In *EDBT*. 433–444.
[36] Shohei Tsuruoka, Daichi Amagata, Shunya Nishio, and Takahiro Hara. 2020. Distributed Spatial-Keyword kNN Monitoring for Location-aware Pub/Sub. In *SIGSPATIAL*. 111–114.
[37] Rares Vernica, Michael J Carey, and Chen Li. 2010. Efficient parallel set-similarity joins using MapReduce. In *SIGMOD*. 495–506.
[38] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering? An adaptive framework for similarity join and search. In *SIGMOD*. 85–96.
[39] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2017. Leveraging set relations in exact set similarity join. *PVLDB* 10, 9 (2017), 925–936.
[40] Xiang Wang, Wenjie Zhang, Ying Zhang, Xuemin Lin, and Zengfeng Huang. 2017. Top-k spatial-keyword publish/subscribe over sliding window. *The VLDB Journal* 26, 3 (2017), 301–326.
[41] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Zengfeng Huang. 2016. Skype: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB* 9, 7 (2016), 588–599.
[42] Dingming Wu, Gao Cong, and Christian S Jensen. 2012. A framework for efficient spatial web object retrieval. *The VLDB Journal* 21, 6 (2012), 797–822.
[43] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S Jensen. 2011. Joint top-k spatial keyword query processing. *IEEE Transactions on Knowledge and Data Engineering* 24, 10 (2011), 1889–1903.
[44] Chuan Xiao, Wei Wang, Xuemin Lin, and Haichuan Shang. 2009. Top-k set similarity joins. In *ICDE*. 916–927.
[45] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Efficient similarity joins for near duplicate detection. In *WWW*. 131–140.
[46] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems* 36, 3 (2011), 1–41.
[47] Jianye Yang, Wenjie Zhang, Xiang Wang, Ying Zhang, and Xuemin Lin. 2020. Distributed Streaming Set Similarity Join. In *ICDE*. 565–576.
[48] Zhong Yang, Bolong Zheng, Guohui Li, Xi Zhao, Xiaofang Zhou, and Christian S Jensen. 2020. Adaptive Top-k Overlap Set Similarity Joins. In *ICDE*. 1081–1092.
[49] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX*. 15–28.
[50] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2016. Inverted linear quadtree: Efficient top k spatial keyword search. *IEEE Transactions on Knowledge and Data Engineering* 28, 7 (2016), 1706–1721.
[51] Kai Zheng, Han Su, Bolong Zheng, Shuo Shang, Jiajie Xu, Jiajun Liu, and Xiaofang Zhou. 2015. Interactive top-k spatial keyword queries. In *ICDE*. 423–434.