Top-k Spatio-Textual Similarity Join

Huiqi Hu, Guoliang Li, Zhifeng Bao, Jianhua Feng, Yongwei Wu, Zhiguo Gong, and Yaoqiang Xu

Abstract—With the development of location-based services (LBS), LBS users are generating more and more spatio-textual data, e.g., checkins and attraction reviews. Since a spatio-textual entity may have different representations, possibly due to GPS deviations or typographical errors, it calls for effective methods to integrate the spatio-textual data from different data sources. In this paper, we study the problem of top-k spatio-textual similarity join (Topk-STJoin), which identifies the k most similar pairs from two spatio-textual data sets. One big challenge in Topk-STJoin is to efficiently identify the top-k similar pairs by considering both textual relevancy and spatial proximity. Traditional join algorithms that consider only one dimension (textual or spatial) are inefficient because they cannot utilize the pruning ability on the other dimension. To address this challenge, we propose a signature-based top-k join framework. We first generate a spatio-textual signature set for each object such that if two objects are in the top-k similar pairs, their signature sets must overlap. With this property, we can prune large numbers of dissimilar pairs without common signatures. We find that the order of accessing the signatures has a significant effect on the performance. So, we compute an upper bound for each signature and propose a best-first accessing method that preferentially accesses signatures with large upper bounds while those pairs with small upper bounds can be pruned. We prove the optimality of our best-first accessing method. Next, we optimize the spatio-textual signatures and propose progressive signatures to further improve the pruning power. Experimental results on real-world datasets show that our algorithm achieves high performance and good scalability, and significantly outperforms baseline approaches.

Index Terms—Similarity join, spatio-textual join, top-k join, spatio-textual signature

1 Introduction

With the rapid development of mobile Internet technology, Internet users are shifting from desktop to mobile devices. Modern mobile devices (e.g., smartphones and tablets) are equipped with GPS, which can help users to easily obtain their locations, and location-based services (LBS) have been widely deployed and well accepted, e.g., Foursquare and Google Map Search. LBS users are generating more and more spatio-textual data which contains both textual descriptions and geographical locations, e.g., checkins and attraction reviews. In user-generated data, a spatio-textual entity may have different representations, possibly due to GPS deviations or typographical errors [3], [18], and it calls for effective methods to integrate the spatio-textual data from different data sources. A spatio-textual similarity join is an important operation in spatio-textual data integration, which, given two sets of spatio-textual objects, finds all *similar* pairs from the two sets, where the similarity can be quantified by combining spatial proximity and textual relevancy (see Section 2.1). There are many applications in spatio-textual similarity joins, e.g., user recommendation in location-based social networks, image duplication detection using spatio-textual tags, spatio-textual advertising, and location-based market analysis [3], [18]. For example, a house rental agency (e.g., rent.com) wants to perform a similarity join on the spatio-textual data of house requirements from renters and the data of house properties from owners. For another example, a startup company, e.g., Factual (factual.com), crawls spatio-textual records to generate points of interest (POIs). As the records are from multiple sources and may contain many duplicates, It needs to run similarity joins to remove the duplicates.

Bouros et al. [3] studied the threshold-based spatio-textual similarity join, which asks users to input a textual threshold and a spatial threshold and identifies the pairs whose textual similarity exceeds the textual threshold and spatial distance is within the spatial threshold. However, in some application, it is rather hard to obtain appropriate thresholds, because a loose threshold involves a large number of answers while a tight threshold generates few results. To address this problem, we study the top-k spatio-textual similarity join (Topk-STJoin) to identify the k most similar pairs, which avoids the process of tuning the thresholds. In the house rental example, the agency has overhead to take renters to show their interested houses. Given a budget (e.g., a limited number of agents who show houses to renters), to maximize the profit, the agency aims to find top-k pairs where the renters in these pairs have large possibilities to rent the corresponding houses. In the duplicate detection example, to remove duplicated POIs, the machine-only algorithms may introduce incorrect results, and a widely-adopted method is to ask the human to check these pairs [13]. However, since the crowd is not free, it is expensive to ask every pair, and thus it aims to select the top-k most similar pairs under a budget.

One big challenge in Topk-STJoin is to efficiently identify the top-k similar pairs by considering both textual

- *H. Hu, G. Li, J. Feng, Y. Wu are with the Department of Computer Science, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing, China. E-mail: hhq11@mails.tsinghua.edu.cn, {liguoliang, fengjh, wuyw}@tsinghua.edu.cn.*
- *Z. Bao is with Computer Science & Info Tech, RMIT University, Australia. E-mail: zhifeng.bao@rmit.edu.au.*
- *Z. Gong is with Department of Computer and Information Science, University of Macau, China. E-mail: fstzgg@umac.mo.*
- *Y. Xu is with East China Grid. E-mail: xu_yq@ec.sgcc.com.cn.*

Manuscript received 21 Mar. 2015; revised 19 Aug. 2015; accepted 19 Sept. 2015. Date of publication 1 Oct. 2015; date of current version 6 Jan. 2016. Recommended for acceptance by K. Chang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2015.2485213

relevancy and spatial proximity. Traditional algorithms that consider only one dimension are rather inefficient because they have no pruning power on the other dimension. To address this challenge, we propose a signature-based top-k similarity join framework. We first extend the prefix-filtering technique for string similarity joins [2], [4] to generate a spatio-textual signature set for each record and utilize them to prune a large number of dissimilar pairs without common signatures. Furthermore, we find that the order of accessing the signatures has a significant effect on the performance. So we study how to access the signatures to achieve high performance. In particular, we make the following contributions.

(1) We propose a signature-based top-k similarity join framework. To the best of our knowledge, this is the first study on top-k spatio-textual similarity join.
(2) We compute an upper bound for each spatio-textual signature, propose a best-first method that preferentially accesses signatures with large upper bounds and prunes those pairs with small bounds, and prove the optimality of our best-first method.
(3) We optimize spatio-textual signatures and propose progressive signatures to improve the pruning power.
(4) Experimental results on two real datasets show that our method significantly outperforms baseline approaches, even by 1-2 orders of magnitude.

The rest of this paper is structured as follows. We define the problem and review related works in Section 2. A signature-based framework is proposed in Section 3. Section 4.2 discusses accessing orders of signatures. We optimize the signatures in Section 5 and make discussions in Section 6. We report results in Section 7 and conclude in Section 8.

2 Preliminaries

2.1 Problem Formulation

Each spatio-textual record $r = \langle r.\mathrm{T}, r.\mathrm{L}\rangle$ includes a spatial location $r.\mathrm{L}$ with latitude and longitude and a textual description $r.\mathrm{T} = \{t_1, t_2 \ldots t_{|r.\mathrm{T}|}\}$ with a set of terms. For simplicity, we interchangeably use $r, r.\mathrm{T}, r.\mathrm{L}$ to denote the record, its term set and its location if the context is clear.

Given two sets of spatio-textual records, a spatio-textual join returns all *similar* records from the two sets. To quantify the similarity between two spatio-textual records, existing methods usually employ a similarity-based metric [3], [18]. First, to measure the textual similarity, we can adopt any textual similarity function, e.g., Jaccard and Cosine. Here we take the well-known Jaccard function as an example and our method can support other textual similarity functions.

Definition 1 (Textual Similarity). *The textual similarity between records r and s is defined as* $\textsc{Sim}_{\mathrm{T}}(r,s) = \frac{|r.\mathrm{T} \cap s.\mathrm{T}|}{|r.\mathrm{T} \cup s.\mathrm{T}|}$, *where $|r.\mathrm{T} \cap s.\mathrm{T}|$ is the size of set $r.\mathrm{T} \cap s.\mathrm{T}$.*

The spatial similarity is evaluated by the spatial distance between two records and is defined as below.

Definition 2 (Spatial Similarity). *The spatial similarity between r and s is defined as* $\textsc{Sim}_{\mathrm{S}}(r,s) = \max\big(0, 1 - \frac{\textsc{Dist}(r.\mathrm{L}, s.\mathrm{L})}{\textsc{Dist}_{max}}\big)$, *where* $\textsc{Dist}(r.\mathrm{L}, s.\mathrm{L})$ *is the distance between $r.\mathrm{L}$ and $s.\mathrm{L}$, and* $\textsc{Dist}_{max}$ *is a user-tolerant distance, which can be set as the maximal distance between any two records.*

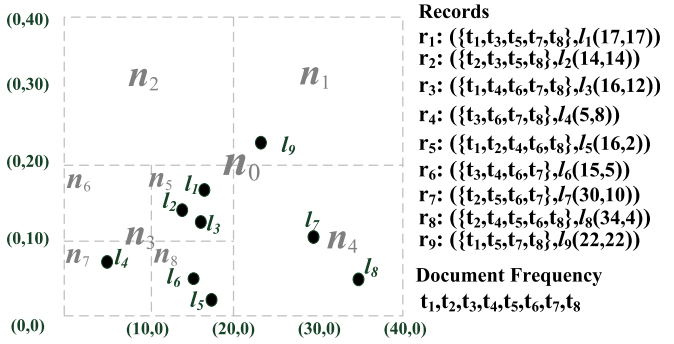


Fig. 1. A running example ($k = 1$, $\alpha = 0.5$).

Then, we combine the textual similarity and spatial similarity to quantify the spatio-textual similarity.

Definition 3 (Spatio-Textual Similarity). *The spatio-textual similarity between r and s is defined as*

$$\textsc{Sim}_{\mathrm{ST}}(r,s) = \alpha \cdot \textsc{Sim}_{\mathrm{T}}(r,s) + (1-\alpha) \cdot \textsc{Sim}_{\mathrm{S}}(r,s),$$

where α is a tuning parameter to leverage the textual relevancy and spatial similarity.

The larger α is, the textual similarity is more important and vice versa. How to select an appropriate value for α has been widely studied [11], [16]. We can utilize existing techniques to set an appropriate value. In this paper we assume α is given. Next, we formally define the top-k spatio-textual similarity join problem.

Definition 4 (Top-k Spatio-Textual Similarity Join). *Given two sets of spatio-textual records $\mathcal{R} = \{r_1, \ldots, r_{|\mathcal{R}|}\}$, $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$, top-$k$ similarity join returns a set $\mathcal{A}$ of k pairs from the two sets with the largest spatio-textual similarities*, i.e., *$\mathcal{A} = \{(r \in \mathcal{R}, s \in \mathcal{S})\}$ such that $\textsc{Sim}_{\mathrm{ST}}(r,s) \geq \textsc{Sim}_{\mathrm{ST}}(r',s')$ for $(r,s) \in \mathcal{A}$ and $(r',s') \in \mathcal{R} \times \mathcal{S} - \mathcal{A}$, and $|\mathcal{A}| = k$.*

We first focus on the self-join problem, i.e., $\mathcal{R} = \mathcal{S}$ and the case of $\mathcal{R} \neq \mathcal{S}$ is discussed in Section 6.

Example 1. Fig. 1 shows nine spatio-textual records, where terms are sorted by document frequencies in an ascending order, i.e., $t_1, \ldots, t_8$. Suppose $\textsc{Dist}_{max} = 40$, $\alpha = 0.5$, $k = 1$. Given a record pair $\langle r_1, r_9\rangle$ with $r_1.\mathrm{T} = \{t_1, t_3, t_5, t_7, t_8\}$, $r_1.\mathrm{L} = [17, 17]$ and $r_9.\mathrm{T} = \{t_1, t_5, t_7, t_8\}$, $r_9.\mathrm{L} = [22, 22]$, $\textsc{Sim}_{\mathrm{T}}(r_1, r_9) = \frac{|\{t_1,t_5,t_7,t_8\}|}{|\{t_1,t_3,t_5,t_7,t_8\}|} = 0.8$, $\textsc{Sim}_{\mathrm{S}}(r_1, r_9) = 1 - \frac{\sqrt{(17-22)^2+(17-22)^2}}{40} = 0.823$ and $\textsc{Sim}_{\mathrm{ST}}(r_1, r_9) = 0.812$. The top-1 answer is $\langle r_1, r_9\rangle$.

2.2 Related Works

Spatio-Textual Similarity Join. To the best of our knowledge, this is the first study on top-k spatio-textual similarity join. There are two works on threshold-based spatio-textual similarity joins [3], [19]. Both require users to input two thresholds, one for textual similarity and the other for spatial similarity, and identify the similar pairs satisfying the two

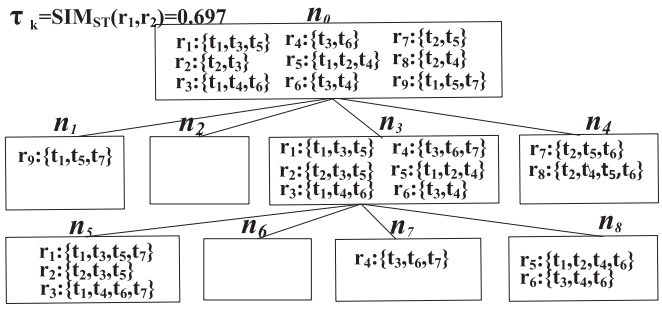


Fig. 2. Example of signatures.

threshold-based constraints. Bouros et al. [3] combined existing string similarity join techniques with grid partitions, and we extend this method to support our problem (see Section 7). However, their method uses separate textual and spatial thresholds and it is hard to find approximate thresholds for the top-k similarity join problem. Liu et al. [18], [19] utilized spatial regions as the spatial part and evaluated the spatial similarity based on the overlap of two spatial regions. This problem is different from ours and cannot be used to address our problem.

Top-k Spatio-Textual Search. Spatial keyword search [5], [6], [9], [15], [17], [22], [26], [29], [30], [31] has been widely studied and some works focused on top-k spatio-textual search [6], [17], [22], [30], [31]. Cong et al. [6] combined inverted lists with R-trees to compute top-k answers. Zhang et al. [30] combined inverted index and Quadtree. We utilize them to support our problem by maintaining the current top-k answers, deducing a bound based on the top-k answers, and pruning dissimilar pairs based on the bound. The method is rather inefficient as it had to find top-k answers for every record.

Top-k Spatial Join. There are some works on top-k spatial join [7], [10], [20], [21], [24]. Corral et al. [7] computed top-k closest pairs in a spatial database. Hjaltason and Samet [10] recursively pushed R-tree nodes into a priority queue from root to leaf based on an upper bound of the minimum distance between two nodes/records. We extend the top-k spatial join algorithms by assuming the maximum textual similarity as one, replacing the upper bound by combining the minimum distance and the maximum textual similarity and utilizing this bound to prune dissimilar pairs.

Top-k Textual Join. Xiao et al. [27] proposed top-k string similarity join which finds the k most similar records using the Jaccard function. They utilized the upper bound of terms to determine the join order. We can also extend this method to support our problem by computing real textual similarity and using one as the spatial bound, and utilizing their sum as an upper bound to prune dissimilar pairs.

Both top-k spatial join and top-k textual join method only filter dissimilar records from one (spatial or textual) dimension, but do not take full advantage of the pruning ability on both dimensions.

Set Similarity Join. There are many works [1], [2], [4], [12], [14], [25], [28] on the set similarity join problem, which finds similar pairs from two sets within a given threshold. Prefix filtering is a widely-adopted technique to address this problem, which computes a prefix for each object such that if two objects are similar, their prefix sets share common signatures [2], [4]. However, prefix filtering only considers the textual dimension. We extend it to support spatio-textual data and devise signature-accessing strategies to support top-k similarity joins. Adaptive prefix filtering [25] was proposed to improve prefix filtering by using multiple signatures. However adaptive prefix filtering is inefficient for top-k joins, and we propose progressive signatures which can address this problem efficiently (see Section 5).

3 A Signature-Based Framework

In this section, we propose a signature-based framework to address the top-k spatio-textual join problem. We first introduce the framework (Section 3.1) and then propose spatio-textual signatures (Section 3.2). Finally we devise a signature-based join algorithm (Section 3.3).

3.1 Framework

To answer top-k queries, we first initialize a priority queue $\mathcal{Q}$ with k randomly selected pairs. Let τ_k denote the smallest similarity of pairs in $\mathcal{Q}$. Obviously we can prune the *dissimilar pairs* whose similarities are smaller than τ_k. Thus we only need to access each pair whose similarity is larger than τ_k, and use such pair to update $\mathcal{Q}$ and τ_k. If the similarities of remaining pairs are smaller than τ_k, we can terminate and the k pairs in $\mathcal{Q}$ are the answers. An important step in this method is to efficiently prune dissimilar pairs and identify the pairs with similarity larger than τ_k. To address this issue, we propose a filter-verification framework. The filter step generates a signature set for each record such that if the similarity of two records is larger than τ_k, their signature sets must share at least one common signature. Based on this property, if two records have no common signature, we prune such pair; otherwise we take the pair as a candidate. The verification step computes the spatio-textual similarity of each candidate. If the similarity is larger than τ_k, we use this candidate pair to update $\mathcal{Q}$ and τ_k. There are two challenges in this framework. The first is to devise effective signatures which will be discussed in Section 3.2. The second is to identify the candidates with common signatures and we propose an efficient algorithm in Section 3.3.

3.2 Spatio-Textual Signatures

Given a threshold τ_k, for each record r, we generate a signature set $\textsc{Sig}(r|\tau_k) = \{\langle t, n\rangle\}$, where t is a term in r and n is a spatial region containing r. Obviously the number of terms contained in r is limited, but the number of regions containing r is infinite. To this end, we utilize hierarchical spatial indices, e.g., Quadtree and R-tree, to control the number of regions containing r. We use Quadtree for illustration purpose and other spatial indexes will be discussed in Section 6. We restrict regions to be tree nodes and the number of nodes containing r can be controlled.

Our objective is to guarantee that if two records r' and r are similar with respect to a threshold τ_k, then $\textsc{Sig}(r|\tau_k) \cap \textsc{Sig}(r'|\tau_k) \neq \phi$. In other words, if $\textsc{Sig}(r|\tau_k) \cap \textsc{Sig}(r'|\tau_k) = \phi$, $\langle r, r'\rangle$ will not be in the top-k answers and we can prune the pair. For example, suppose $\tau_k = \textsc{Sim}_{\textsc{ST}}(r_1, r_2) = 0.697$. Fig. 2 shows the signatures, e.g., $\textsc{Sig}(r_8|\tau_k) = \{\langle t_2, n_4\rangle, \langle t_4, n_4\rangle, \langle t_5, n_4\rangle, \langle t_6, n_4\rangle, \langle t_2, n_0\rangle, \langle t_4, n_0\rangle\}$, $\textsc{Sig}(r_9|\tau_k) = \{\langle t_1, n_1\rangle, \langle t_5, n_1\rangle, \langle t_7, n_1\rangle, \langle t_1, n_0\rangle, \langle t_5, n_0\rangle, \langle t_7, n_0\rangle\}$. As $\textsc{Sig}(r_8|\tau_k) \cap \textsc{Sig}(r_9|\tau_k) =$

ϕ, $\langle r_8, r_9\rangle$ cannot be the top 1 answer. Obviously the smaller the signature set is, the higher the pruning power is. Next we discuss how to generate an effective signature set with the smallest size for a spatio-textual record r.

For any node n containing r, we consider two cases.

(1) n **is a leaf node**. In this case we only need to consider the records in the leaf node n because if a record r' is outside of n, any signature of r' will not contain node n and thus r and r' cannot share a signature with the same node n. It is worth noting that if r' outside n is actually similar to r, they should share a signature $\langle t, n'\rangle$, where n' is an ancestor of n (see the second case). Thus here we only consider record r' in n. Since r and r' are within this "small" leaf node, their spatial similarity should be large. We can estimate the upper bound of their spatial similarity as 1 and thus we get a lower bound of their textual similarity,

$$\mathcal{LB}^{\mathrm{T}}(r|n,\tau_k) = \frac{\tau_k - (1-\alpha)}{\alpha}. \quad (n \text{ is a leaf}) \tag{1}$$

First, consider $\mathcal{LB}^{\mathrm{T}}(r|n,\tau_k) > 0$. Based on Jaccard definition, if r' is similar to r, $|r' \cap r| \geq \mathcal{LB}^{\mathrm{T}}(r|n,\tau_k) \cdot |r' \cup r| \geq \mathcal{LB}^{\mathrm{T}}(r|n,\tau_k) \cdot |r|$. Suppose we sort the terms in all records based on a global order, e.g., their document frequency in ascending order. Based on such order, we define *pivot terms* as below.

Definition 5 (Pivot Term). *Given a threshold τ_k and a record r in a node n, we define r's pivot terms as* $\textsc{Sig}(r|n,\tau_k) = \{t_1, t_2, \ldots, t_{p(r|n,\tau_k)}\}$, *where*

$$p(r|n,\tau_k) = \lfloor |r.\mathrm{T}| \cdot (1 - \mathcal{LB}^{\mathrm{T}}(r|n,\tau_k)) \rfloor + 1. \tag{2}$$

For any record r' in node n, if r' does not contain any pivot term of r, they cannot be similar. The main reason [2], [4] is that there are only $\mathcal{LB}^{\mathrm{T}}(r|n,\tau_k)|r|-1$ non-pivot terms and $\frac{|r' \cap r|}{|r' \cup r|} \leq \frac{\mathcal{LB}^{\mathrm{T}}(r|n,\tau_k)\cdot|r|-1}{|r' \cup r|} < \mathcal{LB}^{\mathrm{T}}(r|n,\tau_k)$. Thus for a leaf node n, $\langle t_i, n\rangle$ is a signature for $i \leq p(r|n,\tau_k)$.

Second, consider $\mathcal{LB}^{\mathrm{T}}(r|n,\tau_k) \leq 0$. Two records may be similar even if they do not share any common term. In this case, we can add a virtual pivot term $*$ and $\langle *, n\rangle$ is a signature.

For example, in Fig. 2, consider the records $\{r_1, r_2, r_3\}$ in the leaf node n_5. For r_1, $\mathcal{LB}^{\mathrm{T}}(r_1|n_5,\tau_k) = \frac{0.697-0.5}{0.5} = 0.394$, $p(r_1|n_5,\tau_k) = \lfloor 5\times(1-0.394)\rfloor+1 = 4$, $\textsc{Sig}(r_1|n_5,\tau_k) = \{t_1, t_3, t_5, t_7\}$. Similarly, $\textsc{Sig}(r_2|n_5,\tau_k) = \{t_2, t_3, t_5\}$ and $\textsc{Sig}(r_3|n_5,\tau_k) = \{t_1, t_4, t_6, t_7\}$. As $\textsc{Sig}(r_2|n_5,\tau_k) \cap \textsc{Sig}(r_3|n_5,\tau_k) = \emptyset$, $\langle r_2, r_3\rangle$ cannot be the top-1 answer.

(2) n **is a non-leaf node.** Suppose node n has c children $n_1, n_2, \ldots, n_c$ and r is in node n_i. Since we have already considered the record pairs in n_i, here we only need to consider the records in $n - n_i = \{n_1, n_2, \ldots, n_{i-1}, n_{i+1}, \ldots, n_c\}$. We first compute the maximal spatial similarity from r to its sibling n_j for $j \neq i$, denoted by $\textsc{MaxSim}_{\mathrm{S}}(r, n_j)$, and

$$\textsc{MaxSim}_{\mathrm{S}}(r, n_j) = 1 - \frac{\textsc{MinDist}(r, n_j)}{\textsc{Dist}_{max}}, \tag{3}$$

where $\textsc{MinDist}(r, n_j)$ is the minimal distance from r to the boundary of n_j which can be computed in $\mathcal{O}(1)$ time. Then we estimate an upper bound of the spatial similarity between r and records in n-n_i as below.

$$\textsc{MaxSim}_{\mathrm{S}}(r, n - n_i) = \max_{j \neq i} \textsc{MaxSim}_{\mathrm{S}}(r, n_j). \tag{4}$$

We get a lower bound of the textual similarity of r,

$$\mathcal{LB}^{\mathrm{T}}(r|n,\tau_k) = \frac{\tau_k - (1-\alpha)\textsc{MaxSim}_{\mathrm{S}}(r, n-n_i)}{\alpha}. \quad (n \text{ is a non-leaf}) \tag{5}$$

Similarly, we define pivot terms of r for node n, i.e., the first $p(r|n,\tau_k) = \lfloor |r.\mathrm{T}| \cdot (1 - \mathcal{LB}^{\mathrm{T}}(r|n,\tau_k)) \rfloor + 1$ terms. If (r, r') is in top-k answers, r and r' share common pivot terms.

For example, consider record r_4 from n_7 which is a child of n_3. We calculate the spatial upper bound of r_4 for n_3 as $\max\{\textsc{MaxSim}_{\mathrm{S}}(r_4, n_5), \textsc{MaxSim}_{\mathrm{S}}(r_4, n_6), \textsc{MaxSim}_{\mathrm{S}}(r_4, n_8)\} = \textsc{MaxSim}_{\mathrm{S}}(r_4, n_6) = 1 - \frac{10-8}{40} = 0.95$. We infer the textual lower bound as $\mathcal{LB}^{\mathrm{T}}(r_4|n_3,\tau_k) = \frac{0.697-0.5\times0.95}{0.5} = 0.444$. As $p(r_4|n_3,\tau_k) = \lfloor 5 \times (1 - 0.444)\rfloor + 1 = 3$, $\textsc{Sig}(r_4|n_3,\tau_k) = \{t_3, t_6, t_7\}$. Similarly, we can generate the signatures for r_5 from another child n_8. As $p(r_5|n_3,\tau_k) = 3$, $\textsc{Sig}(r_5|n_3,\tau_k) = \{t_1, t_2, t_4\}$. As $\textsc{Sig}(r_4|n_3,\tau_k) \cap \textsc{Sig}(r_5|n_3,\tau_k) = \emptyset$, $\langle r_4, r_5\rangle$ is pruned.

Next we define the spatio-textual signature of r.

Definition 6 (Spatio-textual Signature). *Given a threshold τ_k, the signature set of r is* $\textsc{Sig}(r|\tau_k) = \{\langle t, n\rangle\}$, *where $\langle t, n\rangle$ is a signature of r, node n contains r, and t is a pivot term of r w.r.t. n and τ_k.*

Then, we propose the filtering technique: if *a pair (r, r') is in the top-k answers, their signature sets must overlap*. We also prove the minimality of our signature set: *if the signature set is smaller, it misses answers*.

Lemma 1. *Our signature set* $\textsc{Sig}(r|\tau_k)$ *is minimal: if the set is smaller, it misses answers.*

Proof. Given a threshold τ_k, a node n and a record r, suppose its minimum pivot term set is Φ, we prove $\Phi = \textsc{Sig}(r|n,\tau_k)$. We first prove (1) $\Phi \subseteq \textsc{Sig}(r|n,\tau_k)$: for any term $t_i \in \Phi$, we can prove $t_i \in Sig(r|n,k)$ no matter n is a leaf node or non-leaf node based on the definition of pivot term. We then prove (2) $\textsc{Sig}(r|n,\tau_k) \subseteq \Phi$: for any term $t_i \in \textsc{Sig}(r|n,\tau_k)$, suppose there exists a record r' such that $r'.\mathrm{T} = \{t_i, t_{p+1}, \ldots, t_{|r.\mathrm{T}|}\}$, where $p = p(r|n,\tau_k)$ and $\textsc{Dist}(r, r') = 0$, then we have $\textsc{Sim}_{\mathrm{ST}}(r, r') \geq \tau_k$ and $\langle r, r'\rangle$ is an answer. To avoid missing answer, t_i should be a pivot term and $t_i \in \Phi$. Combining (1) and (2), we complete the proof. □

3.3 Signature-Based Algorithm

We devise a signature-based algorithm that utilizes spatio-textual signatures to identify top-k answers. To facilitate identifying the pairs with common signatures (i.e., $\textsc{Sig}(r|\tau_k) \cap \textsc{Sig}(r'|\tau_k) \neq \phi$), we build an inverted index where entries are signatures and each signature is associated with a list of records that contain this signature. We use $\mathcal{L}(\langle t, n\rangle)$ to denote the inverted list of signature $\langle t, n\rangle$. Then if r and r' have a common signature $\langle t, n\rangle$, they both appear in $\mathcal{L}(\langle t, n\rangle)$.

The pseudo-code of the signature-based algorithm is illustrated in Algorithm 1. The algorithm first builds a spatial index using the spatial information of each record

(line 2), randomly selects k record pairs in a same leaf node and puts them into a priority queue $\mathcal{Q}$ and gets a threshold τ_k (line 3), and sorts the terms in each record by their document frequencies in an ascending order (line 4). Next for each record r, it identifies the nodes that contain r by locating its corresponding leaf node (line 6). Then, for each ancestor node n, it computes the corresponding pivot terms based on Equation (2). For each pivot term t, it retrieves the corresponding inverted list $\mathcal{L}(\langle t, n\rangle)$ (line 8). For each record r' on list $\mathcal{L}(\langle t, n\rangle)$, it computes the similarity between r' and r.[1] If their similarity is larger than τ_k, it uses the pair $\langle r, r'\rangle$ to update $\mathcal{Q}$ and τ_k (line 11). Finally, it appends r on $\mathcal{L}(\langle t, n\rangle)$ (line 12).

Algorithm 1. A Signature-Based Algorithm

Input: $\mathcal{R}$: A spatio-textual dataset; k: top-k
Output: $\mathcal{Q}$: Top-k answers

```
1  begin
2    Build a hierarchical tree index on R;
3    Initialize queue Q with k results;
4    Sort terms by df in an ascending order;
5    for each record r do
6      for each identified node n do
7        Compute pivot terms set SIG(r|n, τ_k);
         for each pivot term t ∈ SIG(r|n, τ_k) do
8          Retrieve L(⟨t, n⟩);
9          for each r' on L(⟨t, n⟩) do
10           if SIM_ST(r, r') ≥ τ_k then
11             Update Q/τ_k;
12         L(⟨t, n⟩).APPEND(r);
13     return Q;
14 end
```

Complexity. Each record r has at most $|r|$ pivot terms and is contained in $\mathcal{D}$ tree nodes, where $\mathcal{D}$ is the depth of the spatial index, thus the signature size of each record is $|r|\mathcal{D}$. Each signature is inserted into at most one inverted list. Thus the total space complexity is $\mathcal{O}(\sum_{r\in\mathcal{R}} |r|\mathcal{D})$. The time complexity of accessing the signature is $\mathcal{O}(\sum_{r\in\mathcal{R}} |r|\mathcal{D})$. The complexity to verify candidates is $\mathcal{O}(\sum_{\langle t,n\rangle} c_v|\mathcal{L}(\langle t, n\rangle)|^2)$, where $|\mathcal{L}(\langle t, n\rangle)|$ is the size of $\mathcal{L}(\langle t, n\rangle)$ and c_v is the maximum verification cost (i.e., the maximal term number in a record).

4 Accessing Order

The order of accessing the records has a significant effect on the performance. If we can first access the highly similar record pairs, we can increase the threshold τ_k quickly and prune more dissimilar pairs. For example, consider $\tau_k = \text{SIM}_{\text{ST}}(r_1, r_2) = 0.697$. For r_5 and r_6 from a leaf node n_8, $\text{SIG}(r_5|n_8, \tau_k) = \{t_1, t_2, t_4, t_6\}$ and $\text{SIG}(r_6|n_8, \tau_k) = \{t_3,\ t_4, t_6\}$. $\langle r_5, r_6\rangle$ should be a candidate pair as $\text{SIG}(r_5|n_8, \tau_k) \cap \text{SIG}(r_6|n_8, \tau_k) = \{t_4, t_6\}$. On the other hand, if we first access $\langle r_1, r_9\rangle$ and $\tau_k = \text{SIM}_{\text{ST}}(r_1, r_9) = 0.82$, we can avoid verifying $\langle r_5, r_6\rangle$ as $\text{SIG}(r_5|n_8, \tau_k) = \{t_1, t_2\}$, $\text{SIG}(r_6|n_8, \tau_k) = \{t_3, t_4\}$, and $\text{SIG}(r_5|n_8, \tau_k) \cap \text{SIG}(r_6|n_8, \tau_k) = \emptyset$.

1. If n is a non-leaf node, we only verify r' on list $\mathcal{L}(\langle t, n\rangle)$ that is not in the same child of n with r. We can easily achieve this by keeping the child to which r' belongs in $\mathcal{L}(\langle t, n\rangle)$ and thus can avoid verifying the records from the same children.

To evaluate different accessing orders, we model the records and their signatures as triples $\langle r, t, n\rangle$, where r is a record and $\langle t, n\rangle$ is a signature of r. Our objective is to determine the accessing orders of triples to achieve high performance. Based on the cost complexity in Section 3.3, *an optimal accessing order* is to minimize the total cost. Intuitively, we have three accessing strategies. (1) Textual-first: We access the triples by sorting on term t. Obviously, the term with the smallest frequency has the least probability to match. If two records share infrequent terms, they have large textual similarity. Following this intuition, we access the triples sorted on terms by document frequency (df) in an ascending order. (2) Spatial-first: We access the triples by sorting on node n. Obviously the records in the same deep-level node (e.g., leaf nodes) have large spatial similarity. Thus we want to first access the triples in the deep-level nodes. Following this observation, we access the triples by sorting on nodes in a bottom-up manner. (3) Best-first: The aforementioned two methods sort the triples on one dimension but cannot utilize the other dimension. To achieve high performance, we first access the record with high possibility to be in the top-k answers. Thus we compute a score for each triple by considering both spatial proximity and textual relevancy, and access the triples by sorting on the score. We discuss the details and prove that the best-first accessing order is optimal in Section 4.1.

4.1 Best-First Accessing Order

Given a triple $\langle r, t, n\rangle$, we want to estimate the upper bound of the similarities between r and other records. We first estimate a textual upper bound. For any other record r', we only need to consider the case that r and r' *first match* on t, because if they match on another term t' before t, we can use the triple $\langle r, t', n\rangle$ to find the pair. Let $\text{Pos}(t, r)$ denote the order of t among all terms in r. So there are $\text{Pos}(t, r) - 1$ terms before t in r. The maximal overlap between r and r' given that they first match on t is $|r.\text{T}| - (\text{Pos}(t, r) - 1)$. Thus we have $\text{SIM}_{\text{T}}(r, r') = \frac{|r.\text{T} \cap r'.\text{T}|}{|r.\text{T} \cup r'.\text{T}|} \le \frac{|r.\text{T}|-(\text{Pos}(t,r)-1)}{|r.\text{T}|}$ and can get a textual upper bound.

$$\mathcal{UB}^{\text{T}}(\langle r, t, n\rangle) = \frac{|r.\text{T}| - (\text{Pos}(t, r) - 1)}{|r.\text{T}|}. \tag{6}$$

Obviously for $i < j$, $\mathcal{UB}^{\text{T}}(\langle r, t_i, n\rangle) > \mathcal{UB}^{\text{T}}(\langle r, t_j, n\rangle)$, i.e., the upper bounds for terms in the front are larger than those in the rear. Thus for each record r, $\langle r, t_i, n\rangle$ should be accessed before $\langle r, t_j, n\rangle$ for $i < j$.

Next we estimate a spatial upper bound. We consider two cases. (1) If n is a leaf node, $\mathcal{UB}^{\text{S}}(\langle r, t, n\rangle) = 1$. (2) If n is a non-leaf node: (2.1) If the siblings of n have no spatial overlap with n, e.g., Quadtree and R^+-tree, the minimal distance from r to the four boundaries of n, denoted by $\text{MINDIST}(r, n)$, must be smaller than the distance from r to any record outside of r, thus we can get an upper bound $\mathcal{UB}^{\text{S}}(\langle r, t, n\rangle) = 1 - \frac{\text{MINDIST}(r,n)}{\text{DIST}_{max}}$; (2.2) If the siblings of n have spatial overlap with n, e.g., R-tree, we find the nearest descendants of n without overlap with n's siblings, denoted by n', and compute $\mathcal{UB}^{\text{S}}(\langle r, t, n\rangle) = 1 - \frac{\text{MINDIST}(r,n')}{\text{DIST}_{max}}$; if there is no such descendant, $\mathcal{UB}^{\text{S}}(\langle r, t, n\rangle) = 1$.

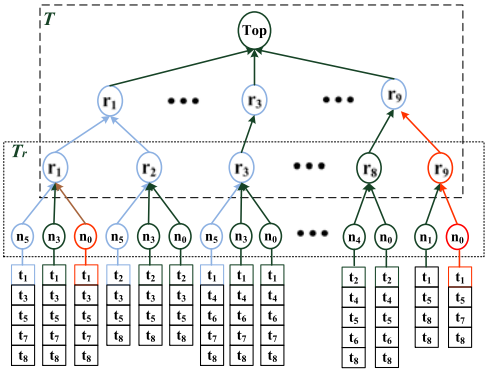


Fig. 3. Example of winner trees.

Next we combine the spatial and textual upper bounds to estimate a spatio-texutal upper bound.

$$\mathcal{UB}^{\mathsf{ST}}(\langle r,t,n\rangle) = \alpha \cdot \mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle) + (1-\alpha)\cdot \mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle). \tag{7}$$

The best-first method accesses the triples $\langle r,t,n\rangle$ sorted by $\mathcal{UB}^{\mathsf{ST}}(\langle r,t,n\rangle)$ in a descending order. Since there are large numbers of triples, it is expensive to directly sort them. To efficiently retrieve the triples in order, we build a 2-layer winner tree [23]. At the bottom level, for each record r, we maintain a winner tree $\mathcal{T}_r$ which is utilized to compute the triple with the largest upper bound for r.[2] At the top level, we build a winner tree $\mathcal{T}$ on top of the record winner trees (using their top triples) to identify the triple with the largest upper bound. The top triple in $\mathcal{T}$ has the largest bound. When popping a triple from $\mathcal{T}$, e.g., $\langle r,t,n\rangle$, we get the next triple from the corresponding bottom-level winner tree, e.g., $\mathcal{T}_r$, and insert this triple into $\mathcal{T}$. Then we adjust $\mathcal{T}$ and compute the next triple with the largest upper bound. Iteratively, we retrieve triples sorted by the score in order.

Example 2. Consider the winner tree in Fig. 3 and records r_1, r_2, r_3 in a leaf node n_5. For the first term t_1 of r_1, we can deduce its textual bound as $\mathcal{UB}^{\mathsf{T}}(\langle r_1,t_1,n_5\rangle) = \frac{5-1+1}{5} = 1$ and its spatial bound is $\mathcal{UB}^{\mathsf{S}}(\langle r_1,t_1,n_5\rangle) = 1$ as n_5 is a leaf node. Thus $\mathcal{UB}^{\mathsf{ST}}(\langle r_1,t_1,n_5\rangle) = 1$. The top winner tree $\mathcal{T}$ will pop $\langle r_1,t_1,n_5\rangle$ as the first triple because it has the largest spatio-textual bound. As $\langle r_1,t_1,n_5\rangle$ is from the bottom level winner tree $\mathcal{T}_{r_1}$, its next triple $\langle r_1,t_3,n_5\rangle$ is added into $\mathcal{T}_{r_1}$. As $\langle r_1,t_3,n_3\rangle$ has the largest bound in $\mathcal{T}_{r_1}$, it is popped from $\mathcal{T}_{r_1}$ and pushed into $\mathcal{T}$. Next the winner tree continues to pop the triples $\langle r_2,t_2,n_5\rangle$ and $\langle r_3,t_1,n_5\rangle$. When accessing $\langle r_3,t_1,n_5\rangle$, $\mathcal{L}(\langle t_1,n_5\rangle) = \{r_1\}$. $\langle r_1,r_3\rangle$ is verified and τ_k is updated as $\tau_k = \textsc{Sim}_{\mathsf{ST}}(r_1,r_3) = 0.651$. For the non-leaf node n_0, the spatial bound $\mathcal{UB}^{\mathsf{S}}(\langle r_1,t_1,n_0\rangle) = 1-\frac{3}{40} = 0.925$. Thus $\mathcal{UB}^{\mathsf{ST}}(\langle r_1,t_1,n_0\rangle) = 0.5\times 1 + 0.5\times 0.925 = 0.963$. As $\mathcal{UB}^{\mathsf{ST}}(\langle r_9,t_1,n_0\rangle) = 0.975 > \mathcal{UB}^{\mathsf{ST}}(\langle r_1,t_1,n_0\rangle)$, $\langle r_9,t_1,n_0\rangle$ will be accessed before $\langle r_1,t_1,n_0\rangle$. When accessing $\langle r_1,t_1,n_0\rangle$, $\mathcal{L}(t_1,n_0) = \{r_5,r_9\}$. $\langle r_1,r_9\rangle$ will be verified and τ_k will be updated as $\tau_k = \textsc{Sim}_{\mathsf{ST}}(r_1,r_9) = 0.812$.

Algorithm 2 shows the pseudo code of the best-first algorithm. In line with Algorithm 1, it still builds a spatial index, initializes $\mathcal{Q}$ and gets a term order (line 2). Then it groups the triples for each record based on nodes, builds a bottom winner tree for each record, and uses the top triples in bottom winner trees to build a top winner tree (line 3). Next it pops the top triple $\langle r,t,n\rangle$ from the top winner tree (line 5). If $\mathcal{UB}^{\mathsf{ST}}(\langle r,t,n\rangle) \le \tau_k$, the algorithm terminates as the upper bound is already smaller than τ_k (line 6). Otherwise, it identifies the inverted list (line 7) and for each record r' on the inverted list, it computes the similarity of r and r'. If their similarity is larger than τ_k, it uses the pair $\langle r,r'\rangle$ to update $\mathcal{Q}$ and τ_k (line 9) and appends the record r on $\mathcal{L}(\langle t,n\rangle)$ (line 10). Next it identifies the triple with the largest upper bound from $\mathcal{T}_r$ (line 11) and inserts it into $\mathcal{T}$ (line 12). Iteratively, the algorithm finds the top-k answers.

Algorithm 2. The Best-First Algorithm

Input: $\mathcal{R}$: A spatio-textual dataset; k: top-k
Output: $\mathcal{Q}$: Top-k Answers

```
1  begin
2    Same to Lines 2-4 in Algorithm 1;
3    Build winner tree T_r and top winner tree T;
4    while T is not empty do
5      ⟨r,t,n⟩ = T.PeekTop();
6      if UB^ST(⟨r,t,n⟩) < τ_k then break;
7      Retrieve L(⟨t,n⟩);
8      for r'∈L(⟨t,n⟩), r',r in different children do
9        if Sim_ST(r,r') ≥ τ_k then Update Q/τ_k;
10     L(⟨t,n⟩).Append(r);
11     ⟨r,t',n'⟩ = T_r.PeekTop();
12     T.Insert(⟨r,t',n'⟩);
13   return Q;
14 end
```

For example, Fig. 4 illustrates the accessing order of the best-first method for computing top-1 answer, where the numbers in square brackets denote the accessing order. $\langle r_1,r_3\rangle$ in $\mathcal{L}(t_1,n_5)$ is the first candidate pair to be verified. $\langle r_1,r_9\rangle$ is verified before some candidates in its child, e.g, $\langle r_2,r_5\rangle$, because the algorithm selects the triples with higher upper bounds first. The algorithm terminates after accessing the $45th$ triple because there is no triple with the upper bound larger than τ_k (the next triple popped out from the winner tree will be $\langle r_1,t_5,n_5\rangle$ with $\mathcal{UB}^{\mathsf{ST}}(\langle r_1,t_5,n_5\rangle) = 0.8 < \tau_k$) while Algorithm 1 accesses 67 signatures. Thus the best-first method can prune many unnecessary signatures.

Next we show that the best-first accessing order is the optimal order in the signature-based framework.

Theorem 1. *The best-first accessing order is the optimal order under the signature-based framework.*

Proof. According to the complexity analysis in Section 3.3, the cost on a signature list($\mathcal{L}(\langle t,n\rangle)$) is determined by (1) the number of generated (accessed) signatures, i.e., the number of inserted records, $c = |\mathcal{L}(\langle t,n\rangle)|$. (2) The number of verifications between records on the list,

2. To facilitate finding the triple with the largest bound in $\mathcal{T}_r$, we group the triples for r by nodes. We keep a sorted triple list for each node where triples are sorted by the upper bounds in a descending order (i.e., the term order from front to rear). Using the first triple of each list, we can get the triple with the largest bound of the record.

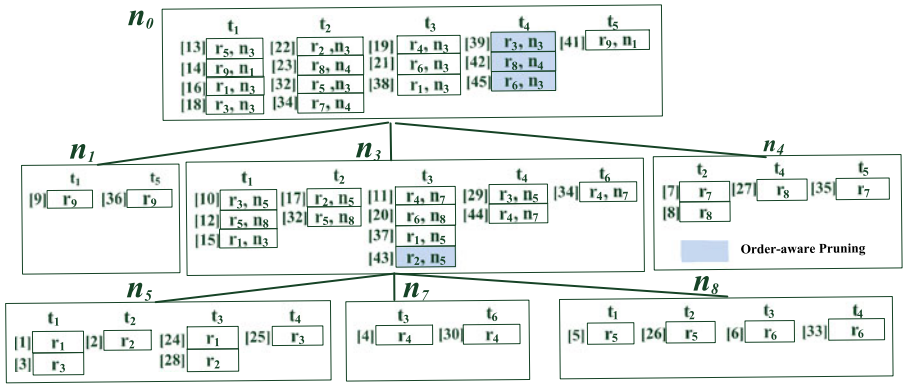


Fig. 4. Example of best-first method.

$\frac{c\cdot(c-1)}{2}$. Next we prove that our method yields the smallest c. We can devise infinite score functions as upper bounds of $\langle r,t,n\rangle$. Let f_0 denote our function (Equation (7)) and $f_1, f_2, \ldots$ denote other functions. Apparently, different functions generate different accessing orders. Let $c_0, c_1, \ldots$ denote the number of corresponding inserted records for $f_0, f_1, \ldots$. Next we prove $c_0 < c_i$.

First, we have three *facts*. (1) The total number of records that can be inserted into the list of $\langle t,n\rangle$ is same, which is independent of any functions and only determined by the dataset, i.e., the document frequency of t among all the records in node n. We denote the total number as C. (2) The real top-k result and τ_k are also independent of the function and $\langle t,n\rangle$, since its depends on the dataset. (3) $f_0(r,t,n) \le f_i(r,t,n)$ $(i \ge 1)$, i.e., our method gives the tightest upper bound for every $\langle t,n\rangle$. This can be proved by the fact we may miss correct top-k answers if we utilize another function which returns a smaller value.

Then, with these three facts, we can conclude that the accessing process with f_0 must terminate earlier than other functions. Consider f_0 and any function f_i. Let X denote the rest records that are not inserted into the list with function f_0, i.e., for any record $r \in X$, $f_0(r,t,n) < \tau_k$. Similarly we denote Y as the rest records of utilizing function f_i. Now consider any arbitrary record $r \in Y$, we have $f_i(r,t,n) < \tau_k$. According to fact (3), we have $f_0(r,t,n) \le f_i(r,t,n) < \tau_k$. Therefore, if $r \in Y$, we have $r \in X$. Thus, we have $Y \subseteq X$. According to fact (1), all the lists have the same total number C, and thus we have $c_0 = C - |X| < c_i = C - |Y|$. □

4.2 Order-Aware Pruning

The best-first method has a good property that the upper bound of triples keeps non-increasing, i.e., the latter triples' upper bounds will not exceed the former triples' bounds. Based on this property, considering the current triple $\langle r,t,n\rangle$ and any triple $\langle r',t,n\rangle$ accessed after $\langle r,t,n\rangle$, we can estimate an upper bound of the similarities between $\langle r,t,n\rangle$ and records accessed after $\langle r,t,n\rangle$, denoted as $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle)$. Obviously if $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle) \le \tau_k$, we do not insert r into $\mathcal{L}(\langle t,r\rangle)$, because the later records will not be similar to r. Thus we can prune many dissimilar pairs. Next we introduce how to compute $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle)$. If n is a non-leaf node, there are two cases.

Case (1). $\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle) \ge \mathcal{UB}^{\mathsf{S}}(\langle r',t,n\rangle)$. We have

$$\begin{aligned}\textsc{Sim}_{\mathsf{S}}(r,r') &\le 1-\frac{\textsc{MinDist}(r,n)+\textsc{MinDist}(r',n)}{\textsc{Dist}_{max}},\\ &= \mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle)+\mathcal{UB}^{\mathsf{S}}(\langle r',t,n\rangle)-1,\\ &\le 2\cdot\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle)-1.\end{aligned}$$

$$\textsc{Sim}_{\mathsf{ST}}(r,r') \le \alpha\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)+(1-\alpha)(2\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle)-1).$$

Case (2). $\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle) < \mathcal{UB}^{\mathsf{S}}(\langle r',t,n\rangle)$. Since $\langle r,t,n\rangle$ is accessed before $\langle r',t,n\rangle$, $\mathcal{UB}^{\mathsf{ST}}(\langle r,t,n\rangle) \ge \mathcal{UB}^{\mathsf{ST}}(\langle r',t,n\rangle)$ and $\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle) > \mathcal{UB}^{\mathsf{T}}(\langle r',t,n\rangle)$.

If $|r.\mathrm{T}|\cdot\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle) \le |r'.\mathrm{T}|\cdot\mathcal{UB}^{\mathsf{T}}(\langle r',t,n\rangle)$, we have

$$\begin{aligned}\textsc{Sim}_{\mathrm{T}}(r,r') &= \frac{|r.\mathrm{T}\cap r'.\mathrm{T}|}{|r.\mathrm{T}\cup r'.\mathrm{T}|}\\ &\le \frac{|r.\mathrm{T}|\cdot\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{|r.\mathrm{T}|+|r.\mathrm{T}|\cdot\frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{\mathcal{UB}^{\mathsf{T}}(\langle r',t,n\rangle)}-|r.\mathrm{T}|\cdot\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}\\ &= \frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{1+\frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{\mathcal{UB}^{\mathsf{T}}(\langle r',t,n\rangle)}-\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)} \le \frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{2-\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}.\end{aligned}$$

Similarly if $|r.\mathrm{T}|\cdot\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle) > |r'.\mathrm{T}|\cdot\mathcal{UB}^{\mathsf{T}}(\langle r',t,n\rangle)$, we also have $\textsc{Sim}_{\mathrm{T}}(r,r') \le \frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{2-\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}$. Thus ,

$$\textsc{Sim}_{\mathsf{ST}}(r,r') \le \alpha\cdot\frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{2-\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}+(1-\alpha)\cdot\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle).$$

Finally, we can compute $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle)$ as below.

$$\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle) = \max\begin{cases}\alpha\cdot\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)+(1-\alpha)\cdot(2\cdot\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle)-1)\\ \alpha\cdot\frac{\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}{2-\mathcal{UB}^{\mathsf{T}}(\langle r,t,n\rangle)}+(1-\alpha)\cdot\mathcal{UB}^{\mathsf{S}}(\langle r,t,n\rangle).\end{cases} \quad (8)$$

Next we design a pruning technique: *for each triple $\langle r,t,n\rangle$, where n is a non-leaf node, if $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle) \le \tau_k$, we do not insert r into $\mathcal{L}(\langle t,n\rangle)$.*

The pseudo code is shown in Algorithm 3. It replaces line 10 in Algorithm 2 with: computing $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle)$ by Equation (8) (line 1); if $\mathcal{UB}^{\mathsf{ST}}_{\mathcal{O}}(\langle r,t,n\rangle) \ge \tau_k$, appending r into the list (line 2).

Algorithm 3. Order-Aware Pruning

```
// replace line 10 in Algorithm 2
```
1 Compute $\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r,t,n\rangle)$ based on Equation (8);
2 **if** $\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r,t,n\rangle) \geq \tau_k$ **then** $\mathcal{L}(\langle t,n\rangle)$.Append($r$);

For example, considering the $39th$ triple $\langle r_3,t_4,n_0\rangle$, we have $\mathcal{UB}^{\mathsf{S}}(\langle r_3,t_4,n_0\rangle) = 1-\frac{4}{40} = 0.9$, $\mathcal{UB}^{\mathsf{T}}(\langle r_3,t_4,n_0\rangle) = 0.8$, thus $\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r_3,t_4,n_0\rangle) = \max\big(0.5\times 0.8 + 0.5\times(1.8-1) = 0.8, 0.5\times\frac{0.8}{2-0.8} + 0.5\times 0.9 = 0.783\big) = 0.8 < \tau_k = 0.812$. As shown in Fig. 4 with blue marks, $\langle r_3,t_4,n_0\rangle$ ($39th$) will not be inserted into $\mathcal{L}(\langle t_4,n_0\rangle)$. Thus, although the triple $\langle r_8,t_4,n_0\rangle$ ($42th$) contains $\langle t_4,n_0\rangle$, $\langle r_8,r_3\rangle$ will not be verified.

5 Progressive Signature

Given a triple $\langle r,t,n\rangle$, for each record r' on $\mathcal{L}(\langle t,n\rangle)$, the signature-based method has to verify $\langle r,r'\rangle$. If there are many records on $\mathcal{L}(\langle t,n\rangle)$, the verification cost will be high. If we can reduce the size of $\mathcal{L}(\langle t,n\rangle)$, we can further improve the performance. To address this issue, we design a progressive signature to improve the pruning power. We first introduce the basic idea (Section 5.1) and discuss how to incorporate progressive signatures into our framework (Section 5.2). Then we devise an efficient algorithm (Section 5.3).

5.1 Basic Idea

Given a record r, a node n, and a threshold τ_k, the signature set of r w.r.t. n and τ_k includes $\langle t_1,n\rangle, \langle t_2,n\rangle, \ldots \langle t_p,n\rangle$, where $p = p(r|n,\tau_k)$. Inspired by adaptive prefix filtering [25], if we select $q-1$ more terms from r, i.e., $t_{p+1}, t_{p+2}, \ldots, t_{p+q-1}$, called *quasi pivot terms*, then if r' is similar to r, they must share at least q pivot terms or quasi pivot terms, as proved in [25]. The basic idea is similar to prefix filtering: even if r' contains all terms after t_{p+q-1} of r, they are still dissimilar. However, adaptive signatures in [11], [25] cannot effectively address our top-k problem, because (1) it aims to reduce the candidate sizes based on a given textual threshold while we focus on top-k join which has no static threshold, and (2) it generates a $(p+q-1)$-length prefix such that two records are similar if their prefixes share at least q common signatures. Then it adopts a count-based method, which scans the inverted list of signatures in the $(p+q-1)$-length prefix of r, counts the occurrence numbers of strings on the inverted lists, and reports the strings with occurrence number exceeding q as candidates. In other words, for each record, it inserts all pivot terms into inverted lists to identify the candidates. However, to incorporate it into our best-first method, we need to assign a spatio-textual bound for each signature. To this end, it has to enumerate $\binom{p+q-1}{q}$ possible cases (requiring to share q common signatures from $p+q-1$ signatures), computes a bound for each case and inserts signatures based on the bounds. Since there are $\binom{p+q-1}{q}$ cases, this method is obviously rather expensive. To address this issue, we propose *progressive signatures*.

Definition 7 (Progressive Signature). *Given a threshold τ_k, the q-length progressive signature set of r is $\textsc{Sig}^q(r|\tau_k) = \{\langle T^q,n\rangle\}$, where $\langle T^q,n\rangle$ is a q-length progressive signature of r, n contains r, and T^q is a q-size subset of $\{t_1,t_2,\ldots,t_{p+q-1}\}$.*

Then we devise a pruning technique: *if two strings are similar, they must share a common q-length progressive signature*. Notice that we will judiciously select several subsets of $\{t_1,\ldots,t_{p+q-1}\}$ as signatures. It is different from the adaptive prefix which equally considers $\binom{p+q-1}{q}$ cases. For example, assume $\tau_k = \textsc{Sim}_{\mathsf{ST}}(r_1,r_9) = 0.812$ and $q=2$. For $\langle r_6,t_3,n_3\rangle$ and $\langle r_1,t_3,n_3\rangle$, $p(r_6|n_3,\tau_k) = 2$ and $p(r_1|n_3,\tau_k) = 2$. Their (quasi) pivot terms are $\{t_3,t_4,t_6\}$ and $\{t_1,t_3,t_5\}$ respectively. We cannot prune $\langle r_1,r_6\rangle$ using the 1-length progressive signature as they share a common signature $\langle t_3,n_3\rangle$. Our method only needs to generate their 2-length progressive signatures for term t_3, i.e., $\{\langle(t_3,t_4),n_3\rangle,\langle(t_3,t_6),n_3\rangle\}$ and $\{\langle(t_3,t_5),n_3\rangle\}$ and can ignore other terms, i.e., $\{t_1,t_4,t_5,t_6\}$, because their 1-length progressive signatures only share t_3 and their 2-length signatures must contain this term. As there is no common 2-length signature, we prune $\langle r_1,r_6\rangle$.

There are two challenges to support progressive signatures. The first is to incorporate progressive signatures into our framework. The second is how to select signatures and decide q. We prefer to select the terms with long lists into signatures in order to improve the pruning ability. To determine q, if q is small, the pruning power is limited (e.g., $q=1$); if q is large, it involves large cost to generate the signatures. To make a tradeoff, we propose effective techniques to judiciously generate signatures.

5.2 Supporting Progressive Signatures

For each triple $\langle r,t,n\rangle$ popped from the top-level winner tree, we first verify $\langle r,r'\rangle$ for each record r' on $\mathcal{L}(\langle t,n\rangle)$ and then check whether $\langle t,n\rangle$ is *valid* (we will discuss the details later). If yes, we still use $\langle t,n\rangle$ and insert r into $\mathcal{L}(\langle t,n\rangle)$; otherwise, we use the 2-length progressive signatures of $\langle t,n\rangle$: $\langle(t,t_{i+1}),n\rangle$, $\langle(t,t_{i+2}),n\rangle$, …, $\langle(t,t_{p+1}),n\rangle$, where $t=t_i$. It is worth noting that for all records that contain the signature $\langle t,n\rangle$, including the records in the current list $\mathcal{L}(\langle t,n\rangle)$ and those accessed after $\langle r,t,n\rangle$, we also need to use their 2-length progressive signatures because $\langle t,n\rangle$ is not a *valid signature*. Thus we split the existing list $\mathcal{L}(\langle t,n\rangle)$ and insert records in $\mathcal{L}(\langle t,n\rangle)$ into inverted lists of 2-length progressive signatures of $\langle t,n\rangle$. Then we use these inverted lists $\mathcal{L}(\langle(t,t_j),n\rangle)$ for $j\in[i+1,p+1]$ (instead of $\mathcal{L}(\langle t,n\rangle)$) to identify candidates (i.e., the pairs on these inverted lists).

There are two challenges in the framework. First, how to decide whether $\langle t,n\rangle$ is valid and whether we should use progressive signatures? We propose a cost-based method. We should compare the cost of two methods: (1) still using $\langle t,n\rangle$ to identify candidates for records accessed after r; and (2) using $\langle(t,t_{i+1}),n\rangle$, $\langle(t,t_{i+2}),n\rangle$, …, $\langle(t,t_{p+1}),n\rangle$ to identify candidates for records accessed after r.

The first method needs to verify (1) the candidate pairs between the records in the current list $\mathcal{L}(\langle t,n\rangle)$ and the records after r and (2) the candidate pairs between records after r. Suppose $\mathcal{N}(\langle t,n\rangle)$ is the number of records in n having $\langle t,n\rangle$ as a signature.[3] The number of candidate pairs is

3. $\mathcal{N}(\langle t,n\rangle)$ is hard to compute. We can utilize the number of records containing t in node n, i.e., $df(t,n)$, to estimate $\mathcal{N}(\langle t,n\rangle)$.

$$|\textsc{Cand}_{q=1}| = \frac{\mathcal{N}(\langle t,n\rangle)(\mathcal{N}(\langle t,n\rangle)-1)}{2} - \frac{|\mathcal{L}(\langle t,n\rangle)|(|\mathcal{L}(\langle t,n\rangle)|-1)}{2}. \quad (9)$$

The verification cost for verifying r and r' is $|r|+|r'|$ which can be estimated by the average term number, denoted by $\texttt{Avg}_t$. Thus the total cost is

$$\textsc{Cost}_{q=1}(\langle t,n\rangle) = 2\texttt{Avg}_t|\textsc{Cand}_{q=1}|. \quad (10)$$

The second method generates 2-length progressive signatures for every record with $\langle t,n\rangle$ as a signature, uses the 2-length progressive signatures to identify candidate pairs for each record after r, and verifies the candidate pairs. The total cost is computed as below.

$$\textsc{Cost}_{q=2}(\langle t,n\rangle) = |\langle t,\widehat{T^2},n\rangle|\mathcal{N}(\langle t,n\rangle) + 2\texttt{Avg}_t|\textsc{Cand}_{q=2}|, \quad (11)$$

where $|\langle r,\widehat{T^2},n\rangle|$ is the average number of 2-length progressive signatures of each record generated from $\langle t,n\rangle$ and $|\textsc{Cand}_{q=2}|$ is the number of candidates using the 2-length progressive signatures. $|\langle r,\widehat{T^2},n\rangle|$ can be estimated based on records in $\mathcal{L}(\langle t,n\rangle)$, i.e.,

$$|\langle r,T^2,n\rangle| \approx \frac{\sum_{r\in\mathcal{L}(\langle t,n\rangle)} p(r|n,\tau_k) - \text{Pos}(t,r)+1}{|\mathcal{L}(\langle t,n\rangle)|},$$

which can be easily computed and materialized when verifying the pairs in $\mathcal{L}(\langle t,n\rangle)$. $\textsc{Cand}_{q=2}$ is a subset of $\textsc{Cand}_{q=1}$ which contains the pairs of records with more than two (quasi) pivot terms. Suppose $|\mathcal{L}^{\geq 2}(\langle t,n\rangle)|$ denotes the number of record pairs with more than two (quasi) pivot terms in $\mathcal{L}(\langle t,n\rangle)$, which can be computed and materialized when verifying the pairs in $\mathcal{L}(\langle t,n\rangle)$. As there are $\frac{|\mathcal{L}(\langle t,n\rangle)|\cdot(|\mathcal{L}(\langle t,n\rangle)-1|)}{2}$ candidates in the current list $\mathcal{L}(\langle t,n\rangle)$ and among them $|\mathcal{L}^{\geq 2}(\langle t,n\rangle)|$ pairs have more than two common (quasi) pivot terms, we can get a ratio. As there are totally $|\textsc{Cand}_{q=1}|$ candidates using 1-length signatures, with the above ratio we estimate $|\textsc{Cand}_{q=2}|$:

$$|\textsc{Cand}_{q=2}| = \frac{|\mathcal{L}^{\geq 2}(\langle t,n\rangle)|}{\frac{|\mathcal{L}(\langle t,n\rangle)|\cdot(|\mathcal{L}(\langle t,n\rangle)-1|)}{2}} \cdot |\textsc{Cand}_{q=1}|. \quad (12)$$

If $\textsc{Cost}_{q=1}(\langle t,n\rangle)) > \textsc{Cost}_{q=2}(\langle t,n\rangle)$, $\langle t,n\rangle$ is not valid anymore. We use the 2-length progressive signatures of $\langle t,n\rangle$; otherwise we still use 1-length signatures. Generally, for an arbitrary signature $\langle T^q,n\rangle (q\geq 2)$, we can also compute the cost for q and $q+1$,

$$\begin{aligned}\textsc{Cost}_q(\langle T^q,n\rangle) &= 2\texttt{Avg}_t|\textsc{Cand}_q|,\\ \textsc{Cost}_{q+1}(\langle T^q,n\rangle) &= |\langle t,\widehat{T^{q+1}},n\rangle|\mathcal{N}(\langle T^q,n\rangle)+2\texttt{Avg}_t|\textsc{Cand}_{q+1}|.\end{aligned} \quad (13)$$

Both $|\textsc{Cand}_q|$ and $|\textsc{Cand}_{q+1}|$ can be similarly computed as Equations (10) and (11). The only difference is that we need to estimate $\mathcal{N}(\langle T^q,n\rangle) \approx \mathcal{N}(\langle T^{q-1},n\rangle \cdot \frac{|\mathcal{L}(T^q,n)|}{|\mathcal{L}(T^{q-1},n)|}$, where T^{q-1} is the subset of T^q by deleting the last term. If $\textsc{Cost}_q(\langle T^q,n\rangle) > \textsc{Cost}_{q+1}(\langle T^{q+1},n\rangle)$, we continue to split the inverted lists of $\langle T^q,n\rangle$ and use $\langle T^{q+1},n\rangle$ as signatures. Otherwise we still use $\langle T^q,n\rangle$ as progressive signatures.

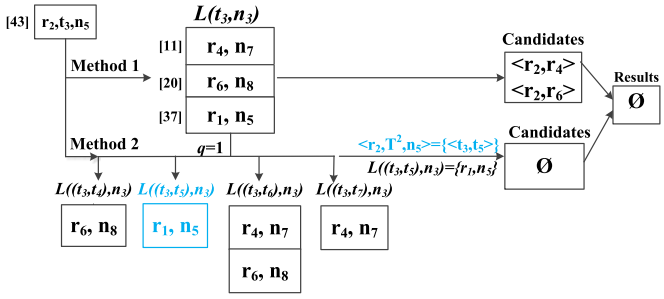


Fig. 5. Example of using progressive signatures.

Example 3. Consider $\mathcal{L}(t_3,n_3)$ in Fig. 5. We assume $\langle r_4,t_3,n_3\rangle$ and $\langle r_6,t_3,n_3\rangle$ have been accessed and $\langle r_1,t_3,n_3\rangle$ is the next triple ($37th$). It accesses $\mathcal{L}(\langle t_3,n_3\rangle)$ and verifies $\langle r_4,r_1\rangle$ and $\langle r_6,r_1\rangle$. Next we estimate the cost of the two methods with Equations (10) and (11). $\mathcal{N}(t_3,n_3)=4$, $\texttt{Avg}_t=4.5$ and $|\mathcal{L}(t_3,n_3)|=3$. Using Equation (9), $|\textsc{Cand}_{q=1}| = \frac{4\times 3}{2} - \frac{3\times 2}{2} = 3$. The cost of the first method is $\textsc{Cost}_{q=1} = 2\times 4.5\times 3 = 27$. For the second method, $|\langle r,\widehat{T^2},n_3\rangle| = \frac{2+2+1}{3} = 1.67$. $|\textsc{Cand}_{q=2}| = \frac{2\cdot|\mathcal{L}^{\geq 2}(t_3,n_3)|}{|\mathcal{L}(t_3,n_3)|\cdot(|\mathcal{L}(t_3,n_3)-1|)} \cdot |\textsc{Cand}_{q=1}| = \frac{2\times 1}{3\times 2}\times 3 = 1$. $\textsc{Cost}_{q=2} = 1.67\times 4+9 = 15.68$. As $\textsc{Cost}_{q=1} > \textsc{Cost}_{q=2}$, we use 2-length progressive signatures. As $\tau_k = \textsc{Sim}_{\text{ST}}(r_1,r_9) = 0.812$, $p(r_4|n_3,\tau_k)=2$ and the 2-length progressive signatures of r_4 generated from $\langle t_3,n_3\rangle$ are $\langle(t_3,t_6),n_3\rangle, \langle(t_3,t_7),n_3\rangle$.

Next we show the pruning power of progressive signatures using the next triple $\langle r_2,t_3,n_3\rangle$ ($43th$). The first method accesses $\mathcal{L}(\langle t_3,n_3\rangle)$ and gets two candidate pairs $\langle r_4,r_2\rangle$ and $\langle r_6,r_2\rangle$ (r_1 is not a candidate as r_1 and r_2 are from the same child of n_3, i.e., n_5). Thus the first method verifies two candidates and checks $7+7$ terms. The second method generates a progressive signature for r_2, $\langle(t_3,t_5),n_3\rangle$. As $\mathcal{L}(\langle(t_3,t_5),n_3\rangle)=\{r_1\}$, $\langle r_4,r_2\rangle$ and $\langle r_6,r_2\rangle$ are pruned. The second method creates progressive signatures for r_4, r_6, r_1, r_2 by scanning $2+2+1+1$ terms and its verification cost is 0 (no candidate). Thus the second method (using progressive signatures) is much better.

The second challenge is how to use the progressive signatures to identify candidates (if is not valid) for a new triple $\langle r,t,n\rangle$. To address this issue, we can use a hash table φ to keep the invalid signature. If $\langle t,n\rangle$ is not in φ, $\langle t,n\rangle$ is still valid and we retrieve the list of $\langle t,n\rangle$; otherwise, we use q-length progressive signatures where $q=\varphi(\langle t,n\rangle)$. *We use an iterative method to generate q-length signatures for r*: first generating 2-length signatures $\langle(t_i,t_{i+1}),n\rangle, \langle(t_i,t_{i+2}),n\rangle, \ldots, \langle(t_i,t_{p+1}),n\rangle$ for $q=2$, $t_i=t$ and then using $(q-1)$-length signatures to generate q-length signatures. If $\langle(t_i,t_{i+j}),n\rangle$ exists in φ (i.e, a previous record has generated this signature), we use $\langle(t_i,t_{i+j}),n\rangle$ to generate $\langle(t_i,t_{i+j},t_{i+j+1}),n\rangle, \ldots, \langle(t_i,t_{i+j},t_{p+2}),n\rangle$ for $q=3$; otherwise, we do not need to extend it and retrieve the inverted lists to identify candidates.

5.3 Progressive-Signature-Based Algorithm

We devise a progressive signature based algorithm similar to Algorithm 2. We first initialize a spatial index, priority queue $\mathcal{Q}$, and the winner trees. Then we peek the triple from the top winner tree. If $\langle t,n\rangle$ is in φ, this signature is not

valid and we generate q-length signatures; otherwise, $\langle t, n\rangle$ is valid, and we still use $\langle t, n\rangle$. For each signature, we retrieve the inverted list, and for each record r' on the list, we verify the pair. Next we compute $\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r, T^q, n\rangle)$ based on Equation (15) and if $\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r, T^q, n\rangle) \geq \tau_k$, we compute Cost_q and Cost_{q+1}. If $\text{Cost}_q > \text{Cost}_{q+1}$, we split the existing list $\mathcal{L}(\langle T^q, n\rangle$, generate the $(q+1)$-length signatures and insert them into inverted lists; otherwise we append the record into current inverted list. Finally we update the winner tree.

Estimate Upper Bound $\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r, T^q, n\rangle)$. Suppose the last term in T^q is t_j. If r' shares a signature $\langle T^q, n\rangle$ with r, they share q terms among the first j terms of r. As there are $|r.\mathrm{T}| - j$ terms after t_j, the textual upper bound is

$$\mathcal{UB}^{\mathsf{T}}(\langle r, T^q, n\rangle) = \frac{q + (|r.\mathrm{T}| - j)}{|r.\mathrm{T}|}. \tag{14}$$

The spatial bound for each term in T^q is the same and thus $\mathcal{UB}^{\mathsf{S}}(\langle r, T^q, n\rangle) = \mathcal{UB}^{\mathsf{S}}(\langle r, t_j, n\rangle)$. So we can get the overall spatio-textual upper bound as below.

$$\mathcal{UB}_{\mathcal{O}}^{\mathsf{ST}}(\langle r, T^q, n\rangle) = \max\begin{cases} \alpha\mathcal{UB}^{\mathsf{T}}(\langle r, T^q, n\rangle) + (1-\alpha)(2 \cdot \mathcal{UB}^{\mathsf{S}}(\langle r, T^q, n\rangle) - 1) \\ \alpha \frac{\mathcal{UB}^{\mathsf{T}}(\langle r, T^q, n\rangle)}{2 - \mathcal{UB}^{\mathsf{T}}(\langle r, T^q, n\rangle)} + (1-\alpha) \cdot \mathcal{UB}^{\mathsf{S}}(\langle r, T^q, n\rangle). \end{cases} \tag{15}$$

6 Discussion

Spatial Index Selection. To evaluate the spatial ability of an index, we sum up the average spatial upper bounds of all the records as the following function.

$$SpatialBound = \sum_{r \in \mathcal{R}} \frac{\sum_{n|r \in n} \textsc{MaxSim}_{\mathrm{S}}(r, n - n_i)}{|n|}, \tag{16}$$

where n is an ancestor of r and $|n|$ is the number of r' ancestors. We aim to select the spatial index with the minimum *SpatialBound*. As we use hierarchical spatial indexes, we only compare two well-known hierarchical indexes, Quadtree and R-tree. Quadtree will be better in our method, because there exist lots of nodes with overlaps in the R-tree index, and for these overlapped nodes, $\textsc{MaxSim}_{\mathrm{S}}(r, n - n_i)$ is equal to the largest value 1. Therefore, the bound of R-tree is looser than that of Quadtree, and accordingly R-tree has larger cost than Quadtree. We also verify this observation by experiments as discussed in Section 7.4.

$\mathcal{R} \neq \mathcal{S}$. We generate the spatio-textual signatures for each dataset. For each $\langle t, n\rangle$, we maintain $\mathcal{L}^{\mathcal{R}}(\langle t, n\rangle)$ for $\mathcal{R}$ and $\mathcal{L}^{\mathcal{S}}(\langle t, n\rangle)$ for $\mathcal{S}$. The pairs on the two inverted lists $(\mathcal{L}^{\mathcal{R}}(\langle t, n\rangle) \times \mathcal{L}^{\mathcal{S}}(\langle t, n\rangle))$ are candidates.

Out-of-Core Setting. A common approach for the case that the dataset cannot be loaded into memory is to partition the dataset into small partitions, use our algorithms to compute the answers on the small partitions, and then combine these answers to generate the final results. Based on this idea, we can devise disk-based or MapReduce-based algorithms [8]. However, in this paper we focus on the in-memory setting and leave devising disk-based or MapReduce-based algorithms as a future work.

TABLE 1
Datasets

Datasets	# Records	Avg # terms	# Distinct terms
Twitter	1 M	17.1	400 K
POI	1 M	52	556 K

7 Experiment

We have conducted extensive experiments to evaluate the efficiency and scalability of our methods.

7.1 Experimental Setup

Datasets. We used two real datasets: Twitter and POI, as shown in Table 1. The Twitter dataset was collected from *twitter.com*, which had 1 million tweets with locations. The POI dataset was crawled from *factual.com*, where the average term number was 5. We merged ten POIs into one record and generated 1 million records.

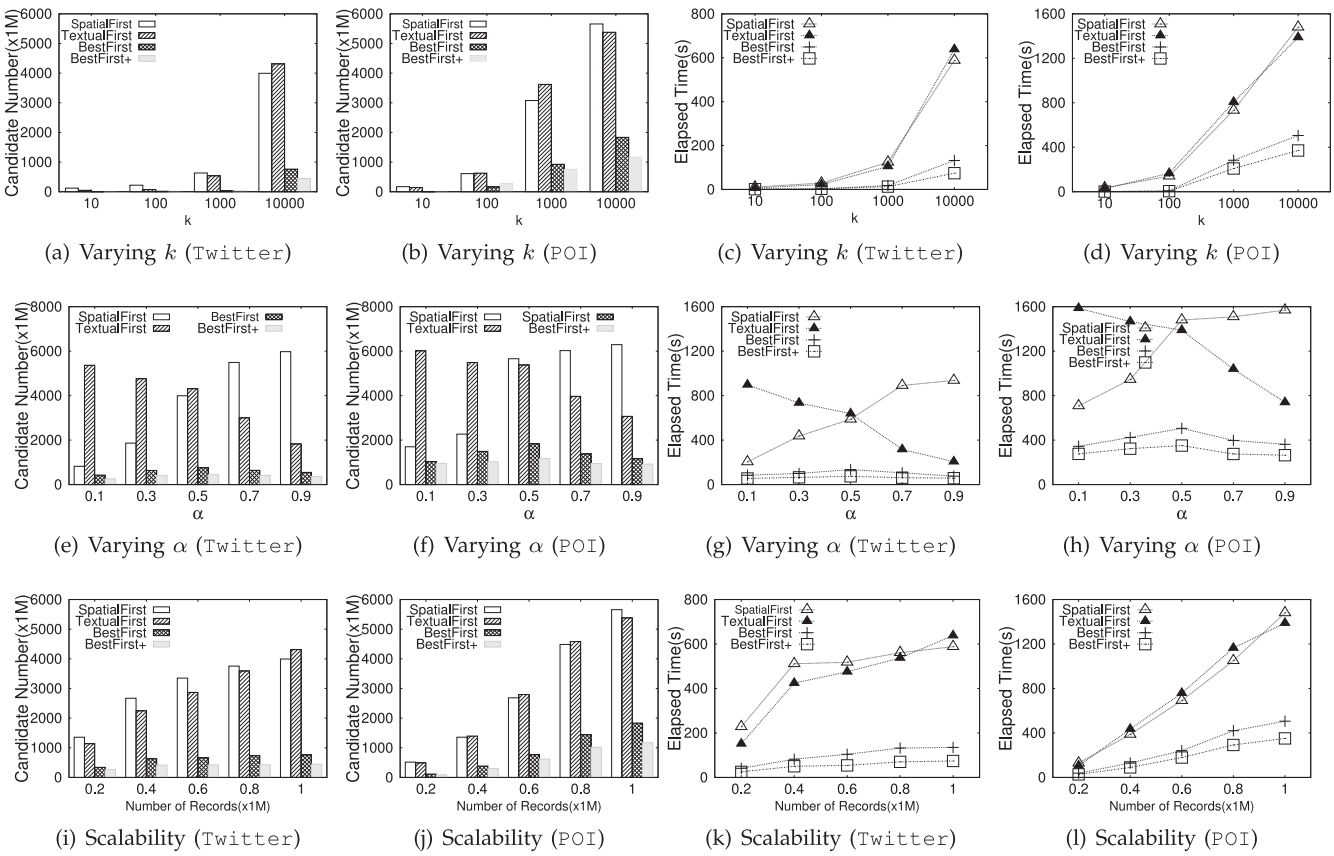
Baselines. We extended five methods to support our problem. (1) Threshold-based spatio-textual join algorithm PPJ-C [3]. As PPJ-C only supported separate spatial and textual thresholds, we extended it with two different strategies. (i) PPJ-C-A: we extended it by decreasing the thresholds with step 0.05 and terminated until getting top-k answers. (ii) PPJ-C-B: we used a priority queue to keep the current top-k answers and deduced a bound. Then we built a hierarchical grid index. For each grid level, we used PPJ-C [3] to compute the results, which joined the eight neighbor grids for each grid. We accessed the grids in a bottom-up manner. Notice that we needed to infer a textual threshold for each grid, which was computed as $\tau_t = \frac{\tau_k - (1-\alpha)\cdot\mathcal{UB}_s(g)}{\alpha}$, where $\mathcal{UB}_s(g)$ is the spatial upper bound of the grid: $\mathcal{UB}_s(g) = 1$ if the grid is a leaf grid; otherwise $\mathcal{UB}_s(g) = 1 - \frac{\textsc{LenSide}(g)}{\textsc{Dist}_{max}}$, where $\textsc{LenSide}(g)$ is the length of the side of g's child grid[4] and $\textsc{Dist}_{max}$ is a user-tolerant distance (see Section 2.1). (2) Top-k spatio-textual search ILQ [30](see Section 2.2). (3) Spatial-first methods, we accessed close records in two manners, (i) SpatialFirst: the bottom-up manner (see Section 4.2). (ii) SpatialFirst-II: the top-down manner [10] (see Section 2.2) (4) Textual-first method TextualFirst [27] (see Section 4.2). As TextualFirst was better than similarity join methods [27], we only compared with TextualFirst.

Setting. All the algorithms were implemented in C++. We used Quadtree for spatial index and inverted lists for textual index for all methods. We terminated to split a Quadtree node if it contained less than 50 K records. We used the in-memory setting. All the experiments were run on a computer with 40 GB RAM, Intel Xeon CPU 2.93 GHz, running Ubuntu.

7.2 Evaluation on Accessing Orders

First, we evaluated the effect of different accessing orders of signatures. We compared four methods, SpatialFirst, TextualFirst, BestFirst and BestFirst+, where BestFirst+

4. Given a grid, PPJ-C computes its results from its eight neighbour grids. As we have computed results from its child level, the minimal spatial distance is the length of side of a grid in its child level.



(a) Varying k (Twitter) (b) Varying k (POI) (c) Varying k (Twitter) (d) Varying k (POI)

(e) Varying α (Twitter) (f) Varying α (POI) (g) Varying α (Twitter) (h) Varying α (POI)

(i) Scalability (Twitter) (j) Scalability (POI) (k) Scalability (Twitter) (l) Scalability (POI)

Fig. 6. Evaluation on accessing orders (default values: $k = 10{,}000$, α=0.5, and $|\mathcal{R}| = 1M$).

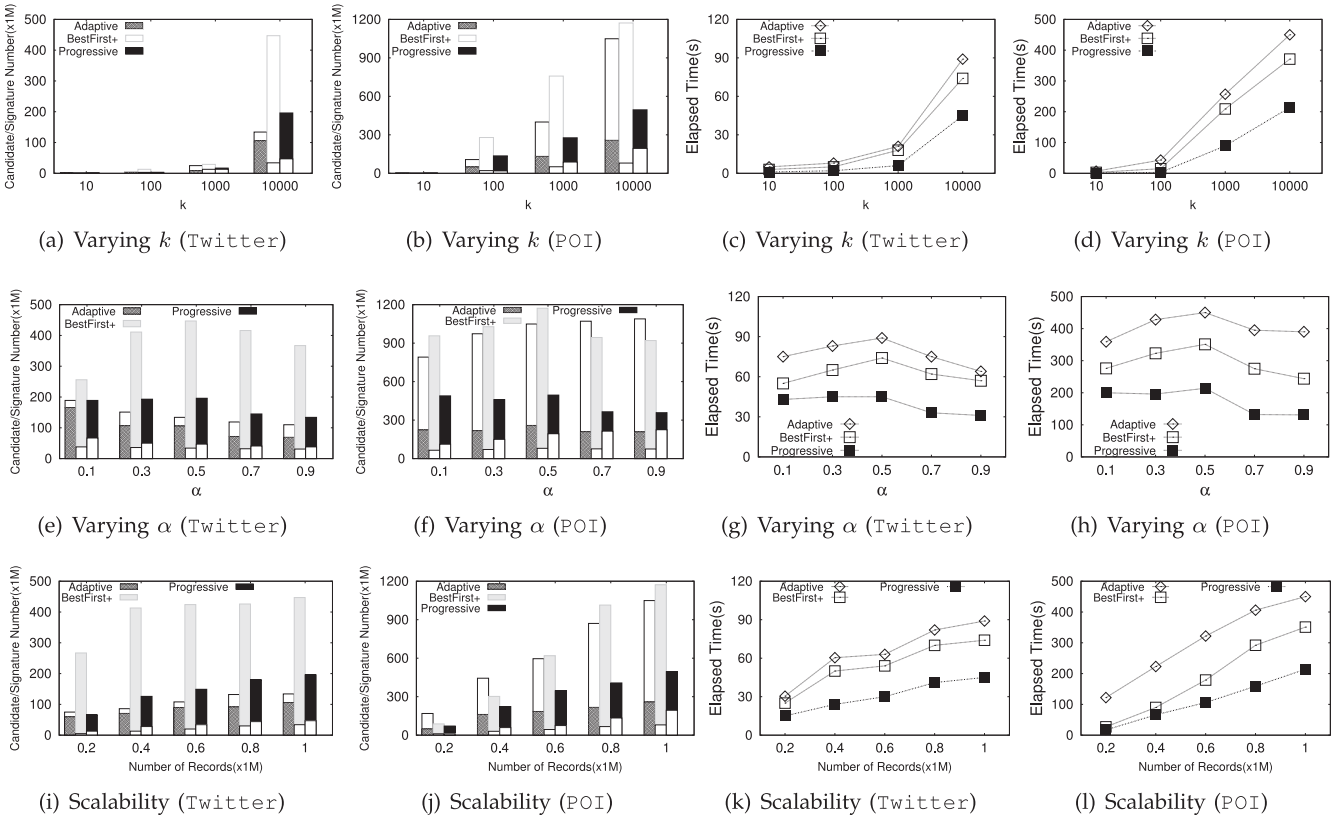
enabled order-aware pruning in BestFirst (Section 4.2). We were aware of three parameters that had impact on the efficiency, i.e., k, α, and the size $|\mathcal{R}|$ of a dataset. When we varied one parameter, others were set as default values: $k = 10{,}000$, $\alpha = 0.5$ and $|\mathcal{R}| = 1M$. We reported the numbers of candidate pairs in and elapsed time in Fig. 6, where the candidates referred to record pairs verified.

We had the following observations. (1) Among all the parameters, BestFirst always outperformed TextualFirst and SpatialFirst because TextualFirst and SpatialFirst accessed signatures with priority on only one dimension, which leaded to many more candidates than BestFirst. For example, for Twitter in Fig. 6i, when $|\mathcal{R}| = 1M$, BestFirst verified $761M$ candidate pairs while TextualFirst and SpatialFirst involved $4{,}000M$ candidates. BestFirst also improved the elapsed time from 600 seconds to 130 seconds as shown in Fig. 6k. (2) BestFirst+ further reduced the candidate number by skipping unnecessary signatures. BestFirst+ outperformed BestFirst by 30-50 percent. (3) With the increase of parameter k, all of these methods generated more candidates and took more time, because a larger k involved more answer pairs which leaded to a small τ_k and thus many pairs cannot be pruned. (4) Parameter α can affect the efficiency of TextualFirst and SpatialFirst. When $\alpha = 0.9$, SpatialFirst took more than 1,000 seconds on POI. This is because SpatialFirst preferred to access pairs with close distance, and a large α indicated the spatial proximity was rather small (even negligible) compared to the textual relevancy. On the contrary, TextualFirst had worse performance when α was small. BestFirst kept stable performance because it considered both the spatial and the textual factors. (5) With the increase of numbers of records, the cost increased because more candidate pairs were generated. However, BestFirst got the slowest growth rate compared to SpatialFirst and TextualFirst. This is attributed to the tighter bounds of BestFirst, which can prune many more dissimilar pairs.

7.3 Evaluation on Progressive Signature

We evaluated our progressive signature Progressive (the method utilizing progressive signatures in Section 5), and compared it with BestFirst+ and Adaptive (extending [25] by enumerating its signature combinations and accessing the combination in order as discussed in Section 5). Fig. 7 shows the efficiency and the candidate number with signature number, where the white (shaded) bars are the number of generated signatures (candidates). We had the following observations. First, Progressive always outperformed BestFirst+ and Adaptive, because progressive signatures can significantly improve the pruning power and generate fewer signatures. For example, when $|\mathcal{R}| = 1M$, BestFirst+ verified 495 millions of pairs on Twitter while Progressive reduced the number to 186 millions. Progressive also reduced the time to 45 seconds from 78 seconds of BestFirst+. Second, although Adaptive can also reduce the number of candidates, it generated large numbers of signatures[5] and it was rather expensive to access large numbers of signatures. For example, when $|\mathcal{R}| = 1M$, Adaptive generated 1,045 millions signatures on POI and got 259 millions of candidates while

5. The cost of generating signatures for Adaptive included retrieving signatures with largest bounds from the winner tree.



(a) Varying k (Twitter) (b) Varying k (POI) (c) Varying k (Twitter) (d) Varying k (POI)

(e) Varying α (Twitter) (f) Varying α (POI) (g) Varying α (Twitter) (h) Varying α (POI)

(i) Scalability (Twitter) (j) Scalability (POI) (k) Scalability (Twitter) (l) Scalability (POI)

Fig. 7. Efficiency evaluation on progressive signatures (default values: $k = 10{,}000$, $\alpha = 0.5$, and $|\mathcal{R}|$=$1M$).

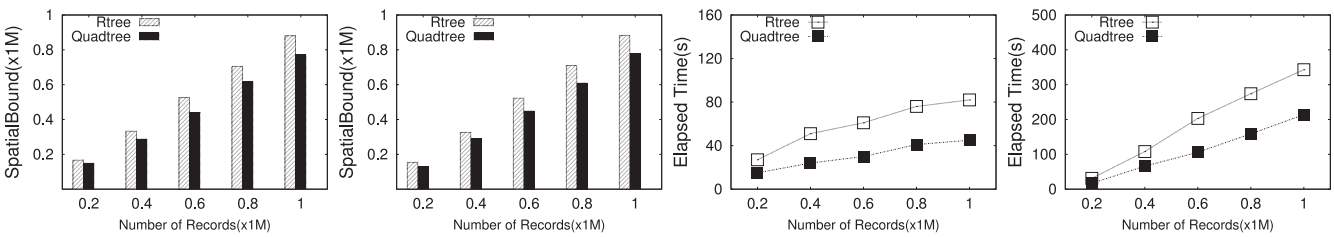
Progressive used 190 millions of signatures and got 495 millions of candidates. Third, with the increase of k, the candidate number and elapsed time of Progressive grew slowly than BestFirst+ and Adaptive. Fourth, different α's have influences on Progressive. Progressive performed better and had larger pruning power for a larger α, because if α was large, the textual similarity was more important and the progressive signatures played a significant role. Fifth, with the increase of dataset sizes, the growth rates of efficiency and the candidate number of Progressive were smaller than BestFirst+ and Adaptive, due to the better pruning power of progressive signatures, which utilized more (quasi) pivot terms to prune dissimilar pairs.

7.4 Comparison of Quadtree and R-Tree

We compared different spatial indexes, Quadtree and R-tree. We used the progressive signature based method, varied the numbers of records, and set k and α as default values. The elapsed time and the spatial bounds (see Equation (16)) are shown in Fig. 8. We can see that Quadtree was twice better than R-tree. This is because Quadtree had much tighter spatial upper bounds with Equation (16), which resulted in less candidates. For example, when $|\mathcal{R}| = 1M$, the average $SpatialBound$ of Quadtree on Twitter was 0.773 (i.e., the average spatial upper bound for each record was 0.773), while the corresponding spatial upper bound for the R-tree index was 0.881. The smaller the value is, the tighter the upper bound is. Thus Quadtree was better than R-tree. Since Quadtree had tighter bounds than R-tree, Quadtree generated fewer signatures and achieved higher performance. For example, Quadtree generated 196 millions candidates and took 45 seconds while R-tree generated 360 millions candidates and took 82 seconds.

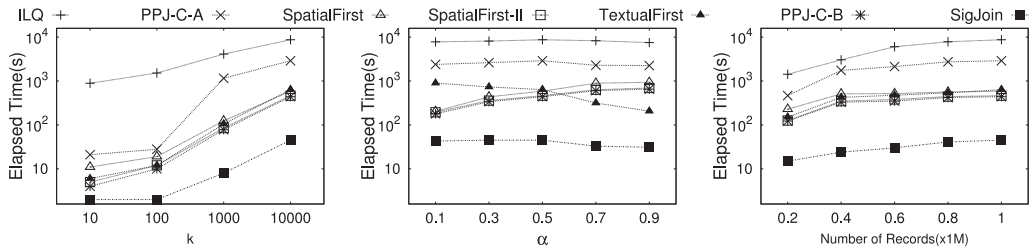
7.5 Comparison with Baselines

We compared our method SigJoin (enabling progressive signatures in BestFirst+) with six baselines ILQ, PPJ-C-A, PPJ-C-B, TextualFirst, SpatialFirst and SpatialFirst-II. Fig. 9 showed the results. We had the following observations. (1) ILQ had the worst performance because ILQ was a

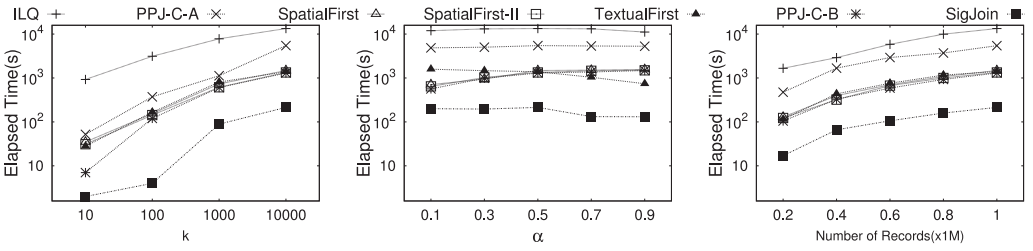


(a) SpatialBound (Twitter) (b) SpatialBound (POI) (c) Elapsed Time (Twitter) (d) Elapsed Time(POI)

Fig. 8. Comparison of Quadtree and Rtree (default values: $k = 10{,}000$, $\alpha = 0.5$).



(a) Varying Parameters on Twitter



(b) Varying Parameters on POI

Fig. 9. Comparison with baselines (default values: $k = 10{,}000$, $\alpha = 0.5$, and $|\mathcal{R}| = 1M$).

TABLE 2
Index Size

Datasets	ILQ	PPJ-C-A	PPJ-C-B	TextualFirst	SpatialFirst	SigJoin
Twitter	1.6 GB	0.9 GB	0.92 GB	0.75 GB	1.3 GB	1.18 GB
POI	4.5 GB	1.49 GB	1.52 GB	1.47 GB	3.8 GB	3.52 GB

search-based method, which needed to scan each record to find top-k answers. (2) PPJ-C-B, TextualFirst and SpatialFirst, SpatialFirst-II and BestFirst+ were better than PPJ-C-A, because it was rather hard for PPJ-C-A to determine appropriate thresholds and PPJ-C-A had to perform multiple similarity join operations for different thresholds and involved many duplicated computations. (3) SpatialFirst, SpatialFirst-II and PPJ-C-B achieved similar performance as all of them verified record pairs based on the priority of spatial distance and PPJ-C-B computed the results in a bottom-up manner using the hierarchical grid index, which was a variant of the spatial-first based method. (4) SigJoin significantly outperformed other methods with 1-2 orders of magnitude, because (a) SpatialFirst, SpatialFirst-II and PPJ-C-B did not optimize the textual pruning and TextualFirst did not optimize the spatial pruning, and they utilized loose upper bounds to find top-k answers, which would verify many pairs of records with large spatial similarities but small textual similarities or with large textual similarities but small spatial similarities. On the other hand, SigJoin chose the verification order by considering both the textual and spatial dimensions, which made τ_k increase most quickly and pruned most dissimilar pairs; (b) SigJoin utilized progressive signatures to enhance the pruning power. (5) When varying parameter α, SigJoin kept a more stable performance than other methods, because SigJoin could progressively estimate a tighter bound to prune dissimilar pairs. (6) With the increase of number of records, SigJoin scaled much better than other methods, because SigJoin estimated a tighter bound and had more powerful pruning.

Index Size. Lastly, we evaluated the space cost of different methods. As illustrated in Table 2, PPJ-C-A and TextualFirst had the least space cost because they did not use any spatial index and this was also why they had poor performance. PPJ-C-B also had small space cost because we cleared up unnecessary indexes of lower-level girds once we moved to the upper levels. SigJoin involved less space than ILQ because ILQ utilized all the terms to create the index while SigJoin used a subset of terms, i.e., pivot terms. SigJoin was better than SpatialFirst because SigJoin accessed smaller numbers of signatures than SpatialFirst, which can save much space cost. The index sizes on POI were larger than those on Twitter for all the methods because POI contained more terms in each record (see Table 1).

8 Conclusion

We proposed a signature-based framework for top-k spatio-textual similarity join. We discussed different accessing orders of signatures proposed a best-first method, and proved the best-first accessing order is optimal. We proposed progressive signatures to improve pruning power. Experimental results showed that our method significantly outperformed baselines.

Acknowledgments

This work was supported by 973 Program of China (2015CB358700), and the NSF of China (61422205, 61472198), YETP0105, Tencent, Huawei, the "NExT Research Center" funded by MDA, Singapore (WBS:R-252-300-001-490), FDCT/116/2013/A3, MYRG105(Y1-L3)-FST13-GZG, National High-Tech R&D (863) Program of China (2012AA012600), and the Chinese Special Project of Science and Technology (2013zx01039-002-002).

REFERENCES

[1] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 918–929.

[2] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 131–140.

[3] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-textual similarity joins," *Proc. VLDB Endownment*, vol. 6, no. 1, pp. 1–12, 2012.

[4] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proc. 22nd Int. Conf. Data Eng.*, 2006, p. 5.

[5] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proc. VLDB Endownment*, vol. 6, no. 3, pp. 217–228, pp. 2013.

[6] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endownment*, vol. 2, no. 1, pp. 337–348, 2009.

[7] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," in *Proc. SIGMOD Int. Conf. Manage. Data*, 2000, pp. 189–200.

[8] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng, "Massjoin: A mapreduce-based method for scalable string similarity joins," in *Proc. Int. Conf. Data Eng.*, 2014, pp. 340–351.

[9] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. 24th Int. Conf. Data Eng.*, 2008, pp. 656–665.

[10] G. R. Hjaltason and H. Samet, "Incremental distance join algorithms for spatial databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1998, pp. 237–248.

[11] H. Hu, Y. Liu, G. Li, J. Feng, and K. lee Tan, "A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 711–722.

[12] Y. Jiang, G. Li, J. Feng, and W. Li, "String similarity joins: An experimental evaluation," *Proc. VLDB Endownment*, vol. 7, no. 8, pp. 625–636, 2014.

[13] G. Lamprianidis, D. Skoutas, G. Papatheodorou, and D. Pfoser, "Extraction, integration and exploration of crowdsourced geospatial content from multiple web sources," in *Proc. 22nd ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2014, pp. 553–556.

[14] G. Li, D. Deng, J. Wang, and J. Feng, "Pass-join: A partition-based method for similarity joins," *Proc. VLDB Endownment*, vol. 5, no. 3, pp. 253–264, 2011.

[15] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 474–485.

[16] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 802–810.

[17] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 585–599, Apr. 2011.

[18] S. Liu, G. Li, and J. Feng, "Star-Join: Spatio-textual similarity join," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 2194–2198.

[19] S. Liu, G. Li, and J. Feng, "A prefix-filter based method for spatio-textual similarity join," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2354–2367, Oct. 2014.

[20] S. Nobari, F. Tauheed, T. Heinis, P. Karras, S. Bressan, and A. Ailamaki, "Touch: In-memory spatial join by hierarchical data-oriented partitioning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 701–712.

[21] S. Qi, P. Bouros, and N. Mamoulis, "Efficient top-k spatial distance joins," in *Proc. 13th Int. Symp.*, 2013, pp. 1–18.

[22] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Proc. 12th Int. Symp.*, 2011, pp. 205–222.

[23] S. Sahni, *Data structures, algorithms, and applications in c++*. Univ. Press, NJ, USA, 1999.

[24] H. Shin, B. Moon, and S. Lee, "Adaptive multi-stage distance join processing," in *Proc. SIGMOD Int. Conf. Manage. Data*, 2000, pp. 343–354.

[25] J. Wang, G. Li, and J. Feng, "Can we beat the prefix filtering?: An adaptive framework for similarity join and search," in *Proc. SIGMOD Int. Conf. Manage. Data*, 2012, pp. 85–96.

[26] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1889–1903, Oct. 2012.

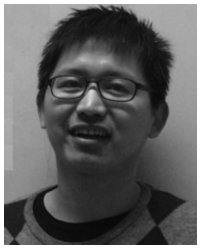
[27] C. Xiao, W. Wang, X. Lin, and H. Shang, "Top-k set similarity joins," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 916–927.

[28] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 131–140.

[29] M. Yu, G. Li, T. Wang, J. Feng, and Z. Gong, "Efficient filtering algorithms for location-aware publish/subscribe," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 4, pp. 950–963, Apr. 2015.

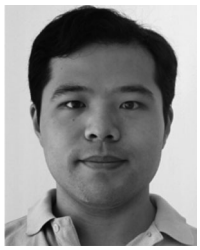
[30] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 901–912.

[31] D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-k spatial keyword search," in *Proc. 16th Int. Conf. Extending Database Technol.*, 2013, pp. 359–370.



Huiqi Hu is currently working toward the PhD degree in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests mainly include spatio-textual data query and spatial database.

Guoliang Li received the PhD degree in computer science from the Tsinghua University, Beijing, China, in 2009. He is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests mainly include data cleaning and integration, spatial databases, and crowdsourcing.

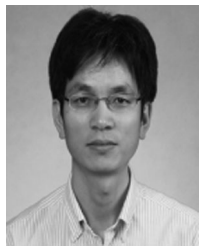


Zhifeng Bao received the PhD degree in computer science from the National University of Singapore in 2011. He is an assistant professor in the RMIT University, Australia. His research interests is generally to build database systems and query models so that they are truly usable, which in particular include exploratory search over social network data, XML data, relational data, and spatial data.

Jianhua Feng received the BS, MS, and PhD degrees in computer science from the Tsinghua University. He is currently working as a professor in the Department of Computer Science, Tsinghua University. His main research interests include large-scale data management and analysis.

Yongwei Wu received the BS and MS degrees from the Lanzhou University, and the PhD degree in Chinese Academy of Science. He is a professor in the Department of Computer Science, Tsinghua University. His research fields include distributed computing, storage systems, and big data analysis.

Zhiguo Gong received the PhD degree from the Department of Computer Science, Institute of Mathematics, Chinese Academy of Science, China. He is an associate professor in Faculty of Science and Technology, University of Macau. His research fields include database systems and web mining.



Yaoqiang Xu received the PhD degree from the Department of Computer Science, Tsinghua University, China. He is an engineer in East China Grid company, Shanghai, China. His research fields include database systems and big data analysis.