Spatio-Textual Similarity Joins[*]

Panagiotis Bouros, Shen Ge, and Nikos Mamoulis
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
{pbouros, sge, nikos}@cs.hku.hk

ABSTRACT

Given a collection of objects that carry both spatial and textual information, a spatio-textual similarity join retrieves the pairs of objects that are spatially close and textually similar. As an example, consider a social network with spatially and textually tagged persons (i.e., their locations and profiles). A useful task (for friendship recommendation) would be to find pairs of persons that are spatially close and their profiles have a large overlap (i.e., they have common interests). Another application is data de-duplication (e.g., finding photographs which are spatially close to each other and high overlap in their descriptive tags). Despite the importance of this operation, there is very little previous work that studies its efficient evaluation and in fact under a different definition; only the best match for each object is identified. In this paper, we combine ideas from state-of-the-art spatial distance join and set similarity join methods and propose efficient algorithms that take into account both spatial and textual constraints. Besides, we propose a batch processing technique which boosts the performance of our approaches. An experimental evaluation using real and synthetic datasets shows that our optimized techniques are orders of magnitude faster than baseline solutions.

1. INTRODUCTION

Databases are becoming increasingly complex over the years, as entities can be easily 'tagged' with different types of auxiliary information, such as keywords and spatial locations. For example, webpages contain keywords and they may also be associated to locations; photographs in photo-sharing services, such as Flickr, are assigned descriptive tags and spatial locations; persons in social networks and customer databases have profile entries (keywords) and addresses. The enrichment of objects with multi-source descriptive information allows for more complex queries and analysis tasks over the data. For example, Flickr offers an API, via which users can search for photos by specifying keywords and a spatial search range. Recently, there has been a growing interest by research and industry to use space as another dimension for organizing and searching text and set-valued data.

[*]Work supported by grant HKU 714212E from Hong Kong RGC.

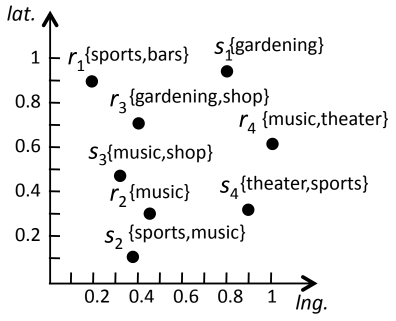


Figure 1: Example of a spatio-textual join

In line with this trend, we investigate the evaluation of *spatio-textual similarity join* (ST-SJOIN) queries; given two collections of objects R and S that carry both spatial and textual information, the ST-SJOIN retrieves the subset J of $R \times S$, such that for every $(r, s) \in J$, r is spatially close to s, based on a distance threshold (i.e., $dist_l(r, s) \leq \epsilon$, where $dist_l$ denotes distance between locations), and the set similarity between r and s also exceeds a threshold θ (i.e., $sim_t(r, s) \geq \theta$, where sim_t denotes textual similarity). Figure 1 illustrates how the join can be used for social recommendations; four men (r_1 to r_4) and four women (s_1 and s_4) are joined based on their locations and interests (shown as keyword sets next to the points). Assuming qualifying pairs should have Euclidean distance $dist_l$ at most $\epsilon = 0.3$ and Jaccard similarity sim_t at least $\theta = 0.5$, the result of the join is $\{(r_2, s_2), (r_2, s_3)\}$.

Applications. ST-SJOIN finds application in a wide range of domains, where spatial and textual information is available for a set of entities. Below, we discuss some examples.

Personal databases. As illustrated in the above example, social networking applications can use the join to identify pairs of people who have similar profiles and they are in nearby locations. The result can be used for *social recommendations*. The join can also serve as a module for *customer segmentation* on a database of customers; the objective is to find groups of people who live nearby and have similar profiles, for directed marketing.

Redundant or dirty data. Data deduplication and cleaning is a classic application of set-similarity joins [10, 16]; originally, in this process, only the textual similarity of data is considered. The spatio-textual similarity join can improve the effectiveness of detecting near-identical entities. For example, consider a database of spatially and textually tagged images (e.g., Flickr). Finding similar image pairs based solely on their tag similarity may not be sufficient, if the tags are not location dependent. Thus, an image tagged as 'bridge' is textually similar to other bridge photos around the world, but can only be actually similar to photographs of the same bridge (taken from nearby locations). A spatio-textual (self) join can be used to identify pairs of images showing the same subject.

Databases with POIs. Applying an ST-SJOIN on a database with points of interest (POIs) can also serve various applications. Pairs of businesses with common themes (e.g., Chinese restaurants) located near each other could collaborate in various ways (e.g., joint advertisement and promotion, location-based market analysis, sharing business processes or inventories). As another example, consider a touristic application, which finds pairs of thematically similar POIs (e.g., archaeological museums), closely located on a map and jointly includes them in targeted tour recommendations.

Contribution. Recently, there has been a lot of work on spatio-textual similarity queries [7, 12, 15, 20, 24]. The input of such a query is a spatial location l and a set of terms K, and the objective is to find objects from a collection R, which are spatially close to l and textually similar to K. In addition, spatial joins [5, 6, 9] and set similarity joins [2, 4, 10, 25, 27, 28] are well-studied problems. However, to our knowledge, there is only one work ([3]) on spatio-textual similarity joins, where in fact the problem is defined differently; given two datasets R and S, the best match in S for each object in R is retrieved. This paper attempts to fill this gap by studying efficient solutions for this interesting query operation. Like previous work on set similarity joins, we mainly focus on the *self-join* (i.e., $R = S$), which is the case for the most representative applications of this query operation. Still, our solutions easily generalize to the general case, where $R \neq S$.

We explore techniques which consider both join thresholds simultaneously during search. In a nutshell, our methods exploit spatial indexing and pruning techniques to reduce the space where the (more expensive) textual similarity predicate needs to be verified; for the latter, they adapt the state-of-the-art algorithm for set-similarity joins [28]. We investigate alternative approaches for spatial pruning, based on a dynamic grid partitioning or a pre-existing spatial index. Besides, we propose a batch processing technique which dynamically partitons the objects into groups based on their spatial locations and textual content and then performs the join at the groups level. This technique greatly improves the performance of all our methods; as we demonstrate, it is orthogonal to the spatial join predicate, since it drastically reduces the cost of the state-of-the-art set-similarity join algorithm [28]. We perform experiments with large-scale real and synthetic datasets, showing that our proposed techniques offer orders of magnitude performance improvement compared to baseline solutions.

Outline. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formally defines the ST-SJOIN operation. Section 4 describes in detail the state-of-the-art set-similarity join algorithm. Our methods and their group-level evaluation are presented in Sections 5 and 6, respectively. Section 7 includes an experimental evaluation and Section 8 concludes the paper.

2. RELATED WORK

Our work is related to spatial distance joins, set-similarity joins, and spatio-textual search. Sections 2.1 to 2.3 summarize related work done in these areas.

2.1 Spatial Distance Joins

The dominant indexing method for spatial data is the R-tree [17], which indexes minimum bounding rectangles (MBRs) of spatial objects hierarchically. Efficient algorithms for spatial intersection joins [6] have been developed for data indexed by R-trees. The ST-SJOIN extends the ϵ-distance join [9]. Given two spatial datasets R and S, the ϵ-distance join finds the pairs (r, s) such that $r \in R$, $s \in S$, and $dist_l(r, s) \leq \epsilon$. The ϵ-distance join can be processed similarly to a spatial intersection join; the R-trees, which index R and S, are concurrently traversed by recursively following pairs of entries for which the MBRs have minimum distance at most ϵ. Techniques for minimizing the distance computation cost between objects and MBRs are proposed in [9]. We discuss details on processing an ϵ-distance join using R-trees in Section 5.2, where we combine this approach with a set-similarity join method.

2.2 Set Similarity Joins

Recently, the set-similarity join has attracted significant interest. Given a collection D of set-valued data, the problem is to find pairs (x, y) of sets in D, such that $sim_t(x, y) \geq \theta$, where $sim_t(\cdot, \cdot)$ is a similarity function and θ is a threshold. The main application of set-similarity joins is near-duplicate object detection [14] (e.g., identify plagiarism, record linkage in data integration, duplicate data cleansing, etc.). Set-similarity joins can also be used to facilitate string matching; [16] showed that the edit distance between two strings can be bounded by set-similarity measures defined on two sets of q-grams, which approximate the strings.

Computing set-similarity joins based on inverted files [31] was first proposed in [25]: for each object x, the inverted lists that correspond to x's elements are scanned to accumulate the similarity between x and all other objects. Several optimizations over this baseline approach are proposed, including scanning only a smaller subset of x's lists and performing a single pass over the data that constructs the inverted index and computes the join result at the same time. Chaudhuri et al. [10] suggested an efficient filter-refinement framework for set-similarity joins, based on the observation that for two sets x, y to satisfy $sim(x, y) \geq t$, a necessary condition is that prefixes of x and y should have at least some minimum overlap. Arasu et al. [2] showed that this prefix-based filtering is just one of the possible summary schemes that one could use as necessary conditions and provided alternative schemes with theoretical bounds on their effectiveness. Bayardo et al., [4] proposed an efficient framework for evaluating set-similarity joins, which minimizes the necessary elements to add in the inverted file, during join evaluation, based on pre-computed bounds on the element weights in the sets and appropriate orderings for the domain of set elements and the database $\mathcal{D}$. This method is further optimized by Xiao et al. [28], by enhancing prefix-filtering using positional information of elements in the prefixes and partially-seen suffixes of the joined objects. In Section 4, we describe the method of [28] in detail, because we use it as a module in our methods and we optimize it in Section 6. The same authors later extended this technique to compute top-k set-similarity joins, i.e., finding k (x, y) pairs in $\mathcal{D}$ with the highest similarity [27].

2.3 Spatio-Textual Search

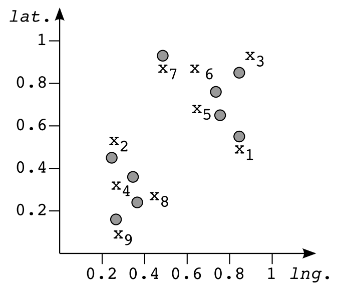
In the past decade, there has been increasing interest on extracting spatial information from web pages such as addresses, phone numbers, zip codes, and then assigning geographic tags to the pages, a process known as *geo-tagging* [1, 13, 21]. Documents are given a geographic *footprint*, i.e., a set of locations; the footprint is often approximated by an MBR. Geo-tagging facilitates multi-criteria search, such as searching documents by textual content and spatial location; this type of search has already been considered by commercial search engines like Google Maps. SPIRIT [26] is a search engine that supports *spatio-textual selection queries*; the user inputs a set of keywords and a set of spatial predicates and the engine returns the documents, which contain the keywords and their spatial footprint satisfies the spatial predicates (e.g., "find all documents about children hospitals within 10km from the city center"). Several indexing approaches for the efficient support of spatio-textual selections have been proposed [11, 18, 26, 30]. Some of these methods propose extensions of the R-tree, which associate nodes or entries of the tree with inverted files for the contents of the corresponding subtrees [18]; other approaches primarily index the data using an inverted file and then spatially index each inverted list by an R-tree [30] or a space-filling curve [11].

De Felipe et al. [15] extend the R-tree to support *containment nearest neighbor* queries. Given a query location q, a set of keywords K, and an integer k, the objective is to find the k nearest objects to q which include all keywords in K. Each entry e of the tree, apart from its MBR, stores a bitmap, which encodes the set of keywords included in every document in the subtree indexed by e. The algorithm of [19] is used to retrieve the nearest neighbors of q incrementally; entries that violate the keyword containment constraint of the query are pruned during search. Cong et al. [12] and Li et al. [20] independently proposed an IR-tree index, which primarily indexes the data using an R-tree, but creates an inverted file for each node of the tree. The inverted file of a leaf node indexes all documents in the node, while in the inverted file of a non-leaf node, each id corresponds to a child (i.e., subtree) of the non-leaf node. The inverted list for a term, contains the children which include that term and the *maximum* weight of the term in any object of the corresponding subtree. By extending the nearest neighbor algorithm of [19], the IR-tree can be used to answer *spatio-textual proximity* queries, where the user provides a location q and a set of keywords K and asks for the best object on a map with respect to both distance from q and similarity with K. An alternative, *Spatial Inverted Index* for spatio-textual proximity queries was recently proposed by [24]. This method generates one inverted list per term and indexes each long inverted list using an aggregate R-tree [23]; given a query, the lists of the query terms are joined, by accessing from each tree the objects in increasing order of relevance and merging them, until the k best objects are guaranteed to be found. Several, more complex queries have also been defined and studied in the context of spatio-textual search, like prestige-based spatio-textual similarity [7] and finding spatially close groups of objects that match the query keywords [8, 29].

To the best of our knowledge, spatio-textual similarity join has been studied only in [3] as SpSJOIN. Compared to our work, SpSJOIN significantly differs from ST-SJOIN. First, similarity between two objects x and y is defined by one measure $sim(x,y) = \frac{sim_t(x,y)}{1+dist_l(x,y)}$ that combines both the spatial distance $dist_l(x,y)$ and the textual similarity $sim_t(x,y)$ of the objects. Second, SpSJOIN is based on a different definition. For each object x it identifies object y that maximizes $sim(x,y)$, i.e, only the best match for x, while ST-SJOIN retrieves all objects y with $dist_l(x,y) \leq \epsilon$, $sim_t(x,y) \geq \theta$. Finally, due to its definition SpSJOIN may return pairs that are of no use, e.g., two photographs from nearby locations that share only one common term on their long descriptions and picture different items.

3. PROBLEM DEFINITION

We define a spatio-textual object x as a triplet $(x.id, x.loc, x.text)$, modeling the identity, the location, and the textual description of x, respectively. The entry $x.loc$ takes values from the two-dimensional geographical space, while $x.text$ is a set of terms drawn from a finite global dictionary $T = \{t_1, t_2, \ldots, t_n\}$. Each term t in $x.text$ could carry a weight (default weights are 1 for unweighted sets), modeling the relevance of t to object x. For example, if x corresponds to a text document, the weights of the terms in $x.text$ could be defined by tf-idf or language models [31]. Finally, we define the *size* $|x|$ of an object x as the number of terms in $x.text$.



x_1: $\{B, C\}$
x_2: $\{E, F\}$
x_3: $\{D, E, F\}$
x_4: $\{A, B, E, F\}$
x_5: $\{C, D, E, F\}$
x_6: $\{C, D, E, F\}$
x_7: $\{A, B, C, D, F\}$
x_8: $\{A, B, D, E, F\}$
x_9: $\{A, B, C, D, E\}$

Figure 2: A collection of spatio-textual objects

For every pair of spatio-textual objects x and y, we define their *spatial distance*, $dist_l(x,y)$, with respect to $x.loc$ and $y.loc$, and their *textual similarity*, $sim_t(x,y)$, as the set similarity between sets $x.text$ and $y.text$, quantified with measures as (weighted) overlap, Jaccard or cosine similarity. The choice of the spatial distance and text similarity measures is highly dependent on the application domain and it is out of the scope of this study. For the rest of the paper, we assume that the spatial distance of objects x and y is the Euclidean distance of their locations, $dist_l(x,y) = dist(x.loc, y.loc)$ and that their textual similarity equals the Jaccard similarity $sim_t(x,y) = \frac{|x.text \bigcap y.text|}{|x.text \bigcup y.text|}$.[1]

Given a collection of spatio-textual objects R, the *spatio-textual similarity join* (ST-SJOIN) identifies pairs of objects in R that are both spatially close and textually similar. Formally, given a spatial distance threshold ϵ and and a textual similarity threshold θ, ST-SJOIN(R, ϵ, θ) retrieves all pairs (x,y) with $x, y \in R$, such that $dist_l(x,y) \leq \epsilon$ and $sim_t(x,y) \geq \theta$.

EXAMPLE 3.1. *Consider the collection of spatio-textual objects $R = \{x_1, \ldots, x_9\}$ in Figure 2.* ST-SJOIN$(R, 0.2, 0.7)$ *retrieves pairs (x_6, x_5), (x_6, x_3) and (x_8, x_4).*

4. BACKGROUND ON SET SIMILARITY JOINS

In this section, we describe in detail the state-of-the-art method for computing textual similarity joins, since our approaches use this method and its filtering mechanisms as modules. An efficient way of computing textual similarity joins w.r.t. a threshold θ, is to use inverted files [31]. An inverted file for a collection of objects R is an index that associates each term t in the global dictionary T to a *postings list* L_t of the objects $x \in R$ that contain t. Assume that such an index is available for R. To compute the join, for each object $x \in R$, we first probe the postings list L_t of every term $t \in x.text$ and accumulate the overlap of x with every involved object $y \in L_t$ in set $O_x[y]$. This way, we identify a set of *candidate pairs* (x,y) for object x. We then fetch each $y \in O_x$ with $O_x[y] > 0$ and compute $sim_t(x,y)$; if $sim_t(x,y) \geq \theta$ then pair (x,y) is added to the join result. An optimization to this approach is to build the inverted index incrementally, while processing the join. For each object x and for each term $t \in x.text$, while scanning L_t to update O_x, we add an entry for x to L_t (to be used by objects following x). This way, every x is only compared with the previously examined objects and each (x,y) pair is considered only once.

[1]Our techniques can easily be adapted for other distance functions and text similarity measures.

4.1 Prefix Filtering

The main problem of the aforementioned simple approach is that the postings lists of the frequent terms can be very long, and therefore, a large number of candidate pairs are generated. Several studies proposed filters to reduce the number of candidate pairs. The most effective technique is based on the *prefix filtering principle* [4, 10]. We first *canonicalize* each object x: we order the terms inside $x.text$ according to a global ordering $\mathcal{O}$, which brings the most infrequent terms in $x.text$ first. Then, we consider a prefix $ppref(x)$ of $x.text$, called *probe prefix*; the length ℓ_p^x of $ppref(x)$ depends on $|x|$ (i.e., the number of terms in $x.text$), the similarity function, and the similarity threshold. In our setting, where Jaccard similarity is considered, and assuming a threshold θ, we have $\ell_p^x = |x| - \lceil \theta \cdot |x| \rceil + 1$. According to the prefix filtering principle, for two objects x and y to be similar, $ppref(x)$ and $ppref(y)$ must share at least one common term. Thus, for each x, in order to obtain the candidate pairs of O_x, we only need to scan and probe the terms contained in $ppref(x)$. This leads to a smaller number of total candidate pairs and significantly, reduces the cost of bookkeeping. For example, consider the collection of objects in Figure 2 and a similarity threshold $\theta = 0.7$. In line with the prefix filtering principle, the candidates set of x_6 with $ppref(x_6) = \{C, D\}$ contains only 2 pairs: (x_6, x_3) and (x_6, x_5).

The `ALL-PAIRS` algorithm [4] builds upon the prefix filtering principle, and in addition, it proposes a way to minimize the size of the inverted index. Specifically, the algorithm examines the objects of the collection in ascending order of their sizes. For each object x, `ALL-PAIRS` probes the L_t postings list of every term $t \in ppref(x)$. Then, due to the order the objects are examined, it only needs to index the terms contained in the so-called *index prefix* of $x.text$, denoted by $ipref(x)$, instead of $ppref(x)$. To clarify this, consider a similarity threshold $\theta = 0.8$, and an object x of size $|x| = 5$ with $ppref(x) = \{A, B\}$. Now, assume another object y with $|y| = 5$ that contains term B but not A in $ppref(y)$. Although we do not know the exact contents of the objects, we can determine their maximum possible overlap $\overline{O}(x, y)$ by adding the number of common terms in their prefixes and the minimum number of unseen terms contained in their suffixes. Specifically, the prefixes of the objects share term B. Further, in the best case the suffix of x would entirely contain suffix of y (since $|x| = |y|$, $|suff(x)| = |suff(y)| = 3$). Thus, the maximum possible overlap $\overline{O}(x, y) = 1 + min(|suff(x)|, |suff(y)|) = 4$ and their Jaccard similarity is at most $\frac{\overline{O}(x,y)}{|x|+|y|-\overline{O}(x,y)} = 0.67$. This signifies that (x, y) is not part of the join result, but most importantly that we do not need to index term B for x. Finally, the length ℓ_i^x of $ipref(x)$ also depends on the size $|x|$ of the object x, the similarity function, and the similarity threshold, and it has at most the length of $ppref(x)$; e.g., in our setting we have $\ell_i^x = |x| - \lceil \frac{2\theta}{1+\theta} \cdot |x| \rceil + 1$.

4.2 The `PPJOIN` Algorithm

The state-of-the-art algorithm for set similarity joins is `PPJOIN` [28].[2] `PPJOIN` extends `ALL-PAIRS`, introducing additionally a *positional* and a *suffix* filter. By *positional information* we refer to the position of a term inside the $x.text$ set of a canonicalized object x. Given two objects x and y, the basic idea of the positional filter is to compute an *upper bound* $\overline{O}$ of their overlap $O(x, y)$, as described in the previous paragraph. If $\overline{O}$ is lower than the minimum overlap of x and y required by the similarity threshold θ, we can safely discard pair (x, y). Finally, regarding the suffix filter, `PPJOIN` operates in a divide-and-conquer manner over the suffixes of objects x and y to compute a *lower bound* $\underline{H}$ of their Hamming distance $H(x, y)$. If $\underline{H}$ is higher than the maximum Hamming distance required for the pair to meet θ, we can safely discard (x, y).

Algorithm 1: `PPJOIN`(R, θ)

```
input  : $R$ is a collection of objects sorted by the increasing order of their
         sizes - each object is canonicalized by a global ordering $\mathcal{O}$; a textual
         similarity threshold $\theta$
output : the set $J$ of all object pairs $(x, y)$, such that $x, y \in R$ and
         $sim_t(x, y) \geq \theta$
1  foreach term $t \in T$ do
2      $L_t \leftarrow \emptyset$
3  foreach object $x \in R$ do
4      $\ell_p^x \leftarrow |x| - \lceil \theta \cdot |x| \rceil + 1$;            // Probe prefix length
5      $\ell_i^x \leftarrow |x| - \lceil \frac{2\theta}{\theta+1} \cdot |x| \rceil + 1$;        // Index prefix length
6      for $pos_x = 1$ to $\ell_p^x$ do
7          $t \leftarrow$ term of $x.text$ at position $pos_x$;
8          foreach entry $\langle y, pos_y \rangle \in L_t$ such that $|y| \geq \theta \cdot |x|$ do
9              if QualifyPositionalFilter($x, pos_x, y, pos_y$) and
                  QualifySuffixFilter($x, pos_x, y, pos_y$) then
10                 $O_x[y] \leftarrow O_x[y] + 1$;       // Increase overlap
11             else
12                 $O_x[y] \leftarrow -\infty$;                 // Prune pair
13         if $pos_x \leq \ell_i^x$ then
14             $L_t \leftarrow L_t \cup \{\langle x, pos_x \rangle\}$;    // Build/extend index
15     Verify($x, O_x, J$);
16 return $J$;
```

Algorithm 1 illustrates the pseudocode of the `PPJOIN` algorithm. `PPJOIN` takes as input a collection of canonicalized objects already sorted in ascending order of their sizes, and a similarity threshold θ. It then iterates through each object x (Lines 3–15). Since all filters are necessary but not sufficient conditions for identifying similar objects, processing an object x involves a filtering and a verification phase. During the filtering phase (Lines 6–14), the algorithm sequentially scans $ppref(x)$, accesses the postings list L_t for each term t in the prefix and defines candidate pairs (x, y) (Lines 8–12). Then, `PPJOIN` filters the pairs using the size filter $|y| \geq \theta \cdot |x|$ proposed in [2] (Line 8), and the positional and suffix filter in Line 9. If a pair (x, y) qualifies all filters, its current overlap is increased and accumulated in $O_x[y]$ (Line 10). Further, in Lines 13–14, the algorithm extents the L_t postings list of every term t in $ipref(x)$. Finally, during verification (Line 15), `PPJOIN` exploits the already accumulated overlap $O_x[y]$ for each candidate pair (x, y) to produce the final results J of the join.

5. SPATIO-TEXTUAL SIMILARITY JOINS

This section presents our methodology for spatio-textual similarity joins which adopts and extends ideas proposed for textual similarity joins. A straightforward way to apply the `PPJOIN` algorithm for computing a ST-SJOIN is to immediately disregard each candidate y which disqualifies the distance constraint with the current object x. This idea is employed by the `PPJ` algorithm presented in Algorithm 2. Compared to `PPJOIN`, the `PPJ` algorithm includes an additional filter in Line 8 to check the spatial distance between two objects x and y. Note that the spatial distance filter is far cheaper than positional and suffix filter, and this is why we apply it first.

EXAMPLE 5.1. *We demonstrate* `PPJ` *for* ST-SJOIN$(R, 0.2, 0.7)$ *over the collection R in Figure 2. Table 1 reports all the iterations performed by the algorithm during the filtering phase. For each object x, we show which pairs $(x, \cdot)$ are considered while probing the postings lists of the index, what* `PPJ` *does with each pair, and finally, how the index is extended. The algorithm checks* 13,

[2]For simplicity, `PPJOIN` denotes the ppjoin+ algorithm [28].

Algorithm 2: PPJ(R, ϵ, θ)

```
input  : $R$ is a collection of spatio-textual objects sorted by the increasing
         order of their sizes - each object is canonicalized by a global ordering
         $\mathcal{O}$; a spatial distance threshold $\epsilon$; a textual similarity threshold $\theta$
output : the set $J$ of all object pairs $(x, y)$, such that $x, y \in R$,
         $dist_l(x, y) \leq \epsilon$ and $sim_t(x, y) \geq \theta$
   ... // Lines 1-7 in Algorithm 1
8  foreach entry $\langle y, pos_y \rangle \in L_t$ such that $dist_l(x, y) \leq \epsilon$ and $|y| \geq \theta \cdot |x|$ do
   ... // Lines 9-16 in Algorithm 1
```

Table 1: Iterations performed during PPJ's filtering phase for computing ST-SJOIN$(R, 0.2, 0.7)$ over collection R in Figure 2 (sf: suffix filter, ϵ: spatial filter)

object	checking pair	indexing
x_1		$L_B = \{\langle x_1, 1\rangle\}$
x_2		$L_E = \{\langle x_2, 1\rangle\}$
x_3		$L_D = \{\langle x_3, 1\rangle\}$
x_4	(x_4, x_1) pruned, ϵ	$L_A = \{\langle x_4, 1\rangle\}$
x_5	(x_5, x_3) pruned, ϵ	$L_C = \{\langle x_5, 1\rangle\}$
x_6	$O_{x_6}[x_5] = 1$ $O_{x_6}[x_3] = 1$	$L_C \bigcup= \{\langle x_6, 1\rangle\}$
x_7	(x_7, x_4) pruned, ϵ (x_7, x_1) pruned, ϵ	$L_A \bigcup= \{\langle x_7, 1\rangle\}$
x_8	$O_{x_8}[x_4] = 1$ (x_8, x_7) pruned, ϵ (x_8, x_1) pruned, ϵ	$L_A \bigcup= \{\langle x_8, 1\rangle\}$
x_9	(x_9, x_4) pruned, ϵ (x_9, x_7) pruned, ϵ (x_9, x_8) pruned, **sf** (x_9, x_1) pruned, ϵ	$L_A \bigcup= \{\langle x_9, 1\rangle\}$

not necessarily distinct, pairs, before proceeding to the verification phase, identifying result pairs (x_6, x_5), (x_6, x_3), *and* (x_8, x_4).

In order to devise more efficient methods than PPJ, we need to exploit the nature of ST-SJOIN. To identify, for every object x in the dataset R, all qualifying (x, y) pairs efficiently, we should exploit the spatial distance threshold ϵ to consider only objects y, for which $dist_l(x, y) \leq \epsilon$. The major drawback of PPJ is that it is unable to examine only these objects; i.e., for each term t contained in the probe prefix of $x.text$, PPJ considers every object y contained in the L_t postings list, no matter how far y is from x in space. This shortcoming of PPJ signifies the necessity for a spatial indexing method that would rapidly prune candidate pairs by their spatial distance. This issue is investigated in Sections 5.1 and 5.2.

5.1 Dynamic Grid Partitioning

Given a query ST-SJOIN(R, ϵ, θ) on a data collection R, we consider a *dynamic grid partitioning* G_R of R's two-dimensional space. The extent of every square cell in the grid in each dimension equals the spatial distance threshold ϵ; hence, the grid is not pre-computed, but dynamically determined by the query parameters. Figure 3(a) illustrates how such a grid can be used. Assume that a probing object x is contained in the grid cell with id 37. To identify candidate (x, y) pairs, we only need to examine the objects contained inside cell 37 itself and its eight adjacent gray cells. In general, at most 9 cells are considered for each object x (e.g., only 6 cells may contain candidates for an object x in cell 2). Note that, the adjacent cells contain a superset of the objects with distance at most ϵ from x; thus, we still need to apply the spatial distance filter for each candidate in these cells. In the example and our implementation, the cells are numbered row-wise from bottom to top.

We now propose two extensions to the PPJ algorithm that exploit this dynamic grid partitioning to speedup the computation of

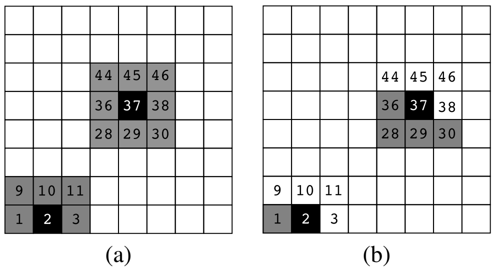


Figure 3: Employing dynamic grid partitioning: (a) PPJ-I defines cell intervals, (b) PPJ-C identifies join cells

ST-SJOIN. Note that for the rest of the paper, we use quadruple $(x.id, x.loc, x.text, x.cid)$ to model a spatio-textual object x, where $x.cid$ is the (dynamically determined) cell-id containing $x.loc$.

5.1.1 *The* PPJ-I *Algorithm*

The PPJ-I algorithm extends PPJ as follows to exploit the dynamic grid partitioning.

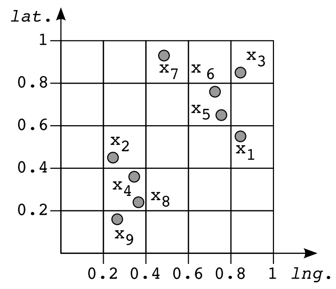
- Before join evaluation, based on the given distance threshold ϵ, PPJ-I defines a grid partitioning of the space. For every grid cell c, PPJ-I identifies at most nine cells adjacent to c and then determines at most three *intervals* of their ids (hence the '-I' in the name of the algorithm). Intuitively, these intervals encode the spatial region around any object contained in cell c. For instance, for cell 37 in Figure 3(a), PPJ-I defines intervals $[28, 30]$, $[36, 38]$ and $[44, 46]$, while for cell 2, only two intervals, $[1, 3]$ and $[9, 11]$, are defined.
- PPJ-I retains the entries $\langle y, pos_y\rangle$ of every postings list in ascending order of $y.cid$. Note that this does not affect correctness as the original PPJOIN algorithm does not impose any ordering for the posting list contents.
- A lightweight *cell-index* on top of every postings list L_t is retained. The cell-index contains an entry $\langle cid, p_{cid}\rangle$ for each cell (cid) which has objects in L_t. p_{cid} is a pointer that provides direct access to the first entry $\langle y, pos_y\rangle$ in L_t that has $y.cid = cid$. In the worst case, the size of the cell-index equals the number of cells in the grid; in practice it is much smaller than the list itself.
- Consider a term t that is contained in the prefix of the probing object x. PPJ-I only accesses the entries $\langle y, pos_y\rangle \in L_t$ such that $y.cid$ falls inside one of the intervals defined for the cell $x.cid$. Thus, PPJ-I first identifies the intervals corresponding to $x.cid$ and then for each interval $i[c_{min}, c_{max}]$ it locates the first posting $\langle y, pos_y\rangle$ in c_{min} using the cell-index on top of L_t; L_t is sequentially scanned from this posting until an entry $\langle y, pos_y\rangle$ with $y.cid > c_{max}$ is found. This approach is reminiscent to spatial query evaluation for data indexed using space-filling curves [22].

Algorithm 3 illustrates the pseudocode of PPJ-I. Since PPJ-I extends PPJ (and PPJOIN), we only detail the changes over Algorithm 1. Specifically, PPJ-I first constructs a dynamic grid partitioning G_R for the input collection of spatio-textual objects R (Lines 1–3); then, for every grid cell c, it defines its intervals and stores them to a set $I[c]$. During the join, for each term t contained in the probe prefix of current object x, the algorithm considers each interval of the cell $x.cid$, accesses the postings list L_t using the cell-index and proceeds similar to PPJ and PPJOIN with identifying and verifying candidate pairs (x, y) (Lines 11–12).

EXAMPLE 5.2. *We demonstrate* PPJ-I *using Example 5.1. First,* PPJ-I *creates a grid partitioning (e.g., 25 cells for* $\epsilon = 0.2$ *in Figure 4). Then, for each accessed object* x *its* $x.cid$ *is dynamically*

Algorithm 3: PPJ-I(R, ϵ, θ)

```
input  : R is a collection of spatio-textual objects sorted by the increasing
         order of their sizes - each object is canonicalized by a global ordering
         $\mathcal{O}$; a spatial distance threshold $\epsilon$; a textual similarity threshold $\theta$
output : the set $J$ of all object pairs $(x, y)$, such that $x, y \in R$,
         $dist_l(x, y) \le \epsilon$ and $sim_t(x, y) \ge \theta$
1  $G_R \leftarrow$ ConstructGridPartitioning$(R, \epsilon)$;
2  foreach cell c in $G_R$ do
3      $I[c] \leftarrow$ DefineCellIntervals$(c, G_R)$;
   ... // Lines 1-7 in Algorithm 1
11 foreach interval $i[c_{min}, c_{max}] \in I[c]$ do
12     foreach entry $\langle y, pos_y \rangle \in L_t$ such that $c_{min} \le y.cid \le c_{max}$ and
       $dist_l(x, y) \le \epsilon$ and $|y| \ge \theta \cdot |x|$ do
           ... // Lines 9-14 in Algorithm 1
19     Verify$(x, O_x, J)$
20 return $J$;
```



$c_2 = \{x_9\}$
$c_7 = \{x_4, x_8\}$
$c_{12} = \{x_2\}$
$c_{15} = \{x_1\}$
$c_{19} = \{x_5, x_6\}$
$c_{23} = \{x_7\}$
$c_{25} = \{x_3\}$

Figure 4: Dynamic grid partition G_R for the collection R in Figure 2, with $\epsilon = 0.2$

determined and for each term $t \in x.text$, the corresponding L_t is searched with the help of the cell-index to find postings in neighboring cells; finally, x is inserted into L_t. During the filtering phase, PPJ-I*, with the help of the grid, considers only 6 out of the 13 pairs checked by* PPJ *in Table 1: (x_5, x_3), (x_6, x_3) for term D, (x_6, x_5) for C and (x_8, x_4), (x_9, x_4) and (x_9, x_8) for term A. For instance, when the current object is x_8 (eighth iteration),* PPJ-I *exploits the fact that x_8 is contained in cell c_7, which has cell intervals $[1, 3]$, $[6, 8]$ and $[11, 13]$. Thus, for term $A \in ppref(x_8)$, when $L_A = \{\langle x_4, 1\rangle, \langle x_7, 1\rangle\}$ is probed, pair (x_8, x_7) is ignored because $x_7.cid = 23$ is not contained in any of the cell intervals.*

5.1.2 The PPJ-C *Algorithm*

Both PPJ and PPJ-I algorithms examine the objects of a collection R in the increasing order of their sizes. In this section, we investigate whether changing this order is possible and if such a change will enhance the computation of ST-SJOIN. For this purpose, we propose another extension to PPJ, termed PPJ-C. The PPJ-C algorithm has the following key features:

- PPJ-C, similar to PPJ-I, defines a dynamic grid partitioning. The objects of R are then examined in ascending order of their cell-id; hence the '-C' in the name of the algorithm. Thus, before commencing the join, PPJ-C orders the objects primarily by cell-id and secondarily by size.
- For each cell c, PPJ-C identifies a set of cells $A[c]$, which contain the objects y that would be joined with every of object x in c. $A[c]$ includes cell c itself, and all cells adjacent to c with smaller ids than c. Thus, each pair of adjacent cells is considered only once. For example, an object x in cell 37 of Figure 3(b) needs to be joined only with the objects contained in the same cell and in the adjacent gray cells, i.e., $A[37] = \{28, 29, 30, 36, 37\}$. Objects belonging to adjacent cells after 37 (e.g., 38) will be joined with x later (e.g., $A[38]$ contains 37).

Algorithm 4: PPJ-C(R, ϵ, θ)

```
input  : R is a collection of spatio-textual objects sorted by the increasing
         order of their sizes - each object is canonicalized by a global ordering
         $\mathcal{O}$; a spatial distance threshold $\epsilon$; a textual similarity threshold $\theta$
output : the set $J$ of all object pairs $(x, y)$, such that $x, y \in R$,
         $dist_l(x, y) \le \epsilon$ and $sim_t(x, y) \ge \theta$
1  $G_R \leftarrow$ ConstructGridPartitioning$(R, \epsilon)$;
2  foreach cell c in $G_R$ do
3      $A[c] \leftarrow$ IdentifyJoinCells$(G_R, c)$;
4      foreach cell $c'$ in $A[c]$ do
5          $J \leftarrow J \cup$ PPJ$(c, c', \epsilon, \theta)$;
6  return $J$;
```

- Unlike PPJ-I (and PPJ), PPJ-C builds an inverted index for every grid cell c, instead of a single global index. Thus, for each term t in the global dictionary there is a $c.L_t$ postings list for every cell c, provided that c contains at least one object x with $t \in ipref(x)$. This scheme results in an advantage over PPJ and PPJ-I regarding the storage requirements, as PPJ-C discards a cell c and its contents after all cells c', for which $A[c']$ includes c, are examined, while PPJ-I keeps all accessed data indexed until the algorithm terminates.
- Consider the currently examined cell c. To compute ST-SJOIN, PPJ-C suffices to perform a self join, for cell c, and at most four non-self joins, i.e., c with every cell $c' \in A[c]$. The self join is evaluated by directly calling PPJ(c, ϵ, θ), which examines the contained objects of c, in the increasing order of their sizes. For a non-self join, [28] suggests merging the contents of the joined sets, determining a global ordering, and then using PPJ again (ignoring pairs that come from the same set). Instead, to join two cells c and c' in PPJ-C, we exploit the size-based ordering of the contents of each cell, and adopt a merge-sort strategy that identifies, at each iteration, the object x with the smallest size between cells c and c'. If $x \in c$ ($x \in c'$) then we probe the inverted index of cell c' (c) and insert x in the inverted index of c (c'). We denote the extension to PPJ that operates on two cells by PPJ$(c, c', \epsilon, \theta)$.

Algorithm 4 illustrates the pseudocode of the PPJ-C algorithm. In Line 1, PPJ-C constructs a dynamic grid partition G_R for the input relation of spatio-textual objects R, similar to PPJ-I. The objects are then ordered according to cell-id and size. Then, in Lines 2–5 the algorithm iterates through the cells of the grid to compute the join. For each cell c, it identifies the $A[c]$ set. For every cell $c' \in A[c]$, PPJ$(c, c', \epsilon, \theta)$ is invoked (this includes the self join PPJ(c, ϵ, θ), since $c \in A[c]$). Note that the objects in each cell c are accessed multiple times (once for every cell c', such that $c \in A[c']$). Thus, PPJ-C buffers the contents of cells that will be needed in later iterations. In practice, we combine the execution of the self join with one the four non-self joins to speed up the ST-SJOIN; i.e., when joining the current cell c with the first $c' \neq c$, $c' \in A[c]$, each object $x \in c$ is probed against the index of c (for the self join) and the index of c' (for the non-self join).

EXAMPLE 5.3. *We demonstrate* PPJ-C *using Example 5.1. Table 2 reports all the iterations performed by* PPJ-C *during the filtering phase, showing the cell joins at each iteration, and for each examined object x the pairs $(x, \cdot)$ considered. The cell joins where no action takes place are omitted. Note that the objects are examined in a different order, compared to* PPJ *and* PPJ-I*. In addition, some objects are examined more than once (e.g., x_9 for the self join of $c_2 \bowtie c_2$ and the non-self join of $c_7 \bowtie c_2$); however, they are probed against smaller lists. Although* PPJ-C *and* PPJ-I *check the*

Table 2: **Iterations performed by PPJ-C during the filtering phase for computing ST-SJOIN$(R, 0.2, 0.7)$ over collection R in Figure 2 (sf: suffix filter, ϵ: spatial filter)**

cell join	object	checking pair	indexing
$c_2 \bowtie c_2$	x_9		$c_2.L_A = \{\langle x_9, 1\rangle\}$
$c_7 \bowtie c_7, c_2$	x_4		$c_7.L_A = \{\langle x_4, 1\rangle\}$
	x_8	$O_{x_8}[x_4] = 1$	$c_7.L_A \bigcup= \{\langle x_8, 1\rangle\}$
	x_9	(x_9, x_4) pruned, ϵ	
		(x_9, x_8) pruned, **sf**	
$c_{12} \bowtie c_{12}, c_7$	x_2		$c_{12}.L_E = \{\langle x_2, 1\rangle\}$
$c_{15} \bowtie c_{15}$	x_1		$c_{15}.L_B = \{\langle x_1, 1\rangle\}$
$c_{19} \bowtie c_{19}, c_{15}$	x_1		
	x_5		$c_{19}.L_C = \{\langle x_5, 1\rangle\}$
	x_6	$O_{x_6}[x_5] = 1$	$c_{19}.L_C \bigcup= \{\langle x_6, 1\rangle\}$
$c_{23} \bowtie c_{23}, c_{19}$	x_5, x_6		
	x_7		$c_{23}.L_A = \{\langle x_7, 1\rangle\}$
$c_{25} \bowtie c_{25}, c_{19}$	x_3		$c_{25}.L_D \bigcup= \{\langle x_3, 1\rangle\}$
	x_5	(x_5, x_3) pruned, ϵ	
	x_6	$O_{x_6}[x_3] = 1$	

same number of object pairs (6 instead of 13 pairs checked by PPJ*), the maximum sizes of their indexes are 4 and 9 entries, respectively.*

5.2 Using an R-tree and the PPJ-R Algorithm

Finally, we investigate how a spatial access method can facilitate the efficient processing of an ST-SJOIN. We assume that the objects are spatially indexed by an R-tree [17] T_R and propose PPJ-R, a method which extends PPJ to apply on the spatial index. The algorithm primarily operates as a spatial ϵ-distance join. Specifically, it takes as input two nodes N_x and N_y of the R-tree (in the first call, $N_x = N_y$ is the root of the tree). If N_x and N_y are non-leaf nodes, then PPJ-R identifies every pair of entries $(e_x, e_y) \in N_x \times N_y$, such that the minimum distance between $MBR(e_x)$ and $MBR(e_y)$ is at most equal to the spatial distance threshold ϵ of ST-SJOIN. These entries may lead to object pairs that qualify the spatial constraint of the join, and therefore, the algorithm runs recursively for the pair of nodes pointed by (e_x, e_y). If N_x and N_y are leaf nodes, PPJ-R applies PPJ to join them (if N_x and N_y are the same, the operation is a self-PPJ).

The algorithm also employs all the optimization techniques proposed for efficiently computing spatial ϵ-distance joins [6, 9]. Specifically, in Lines 3–5, both $MBR(N_x)$ and $MBR(N_y)$ are extended by ϵ in all dimensions and directions (function $E_\epsilon(\cdot)$), and their intersection area A is computed. Every entry $e_x \in N_x$ (resp. $e_y \in N_y$) with $MBR(e_x)$ (resp. $MBR(e_y)$) not intersecting A is discarded. Then, in Lines 6–7, the remaining entries are sorted according to their lower x-dimension bounds (entries in N_y are first extended by ϵ), and finally, their intersection join, denoted by I, is computed using a plane-sweep based heuristic [6]; I contains all pairs (e_x, e_y) such that $e_x \in N_x$, $e_y \in N_y$, and the distance between them in each dimension is equal or less than ϵ.

Note that this method is very similar to PPJ-C, except that (i) the spatial partitions that are joined are determined by the structure of the R-tree and not by the dynamic grid partitioning (ii) the R-tree is used to find the pairs of spatial partitions that should be joined using PPJ. Thus, the extents of the leaf-based partitions are fixed for any value of ϵ; they are governed by the data distribution and the tree parameters (i.e., block size).[3] In practice, this means that PPJ-R is less flexible in determining the partitions, but it does not have the overhead of dynamically defining the grid and determining the cell-id for each object.

[3]Note here that we assume that the tree has been precomputed and is being used by PPJ-R and not dynamically constructed based on the join parameters.

Algorithm 5: PPJ-R$(T_R, N_x, N_y, \epsilon, \theta)$

```
input  : an R-tree $T_R$ indexing a collection of objects $R$; two nodes $N_x$ and
         $N_y$ at the same level of $T_R$; a spatial distance threshold $\epsilon$; a textual
         similarity threshold $\theta$
output : the set $J$ of all object pairs $(x, y)$, such that $x, y \in R$,
         $dist_l(x, y) \leq \epsilon$ and $sim_t(x, y) \geq \theta$
1  if $N_x$ and $N_y$ are leaf nodes then
2      $J \leftarrow J \cup$ PPJ$(N_x, N_y, \epsilon, \theta)$;
3  $A \leftarrow$ MBR$(E_\epsilon(N_x)) \cap$ MBR$(E_\epsilon(N_y))$;          // Space Restriction
4  $M_x = \{$MBR$(e_x) \mid (e_x \in N_y) \wedge ($MBR$(e_x) \cap A \neq \emptyset)\}$;
5  $M_y = \{E_\epsilon($MBR$(e_y)) \mid (e_y \in N_y) \wedge ($MBR$(e_y) \cap A \neq \emptyset)\}$;
6  Sort$(M_x)$; Sort$(M_y)$;
7  $I \leftarrow$ PlaneSweepIntersectionTest$(M_x, M_y)$;   // Plane Sweep
8  foreach pair of entries $(e_x, e_y) \in I$ such that $dist_l(e_x, e_y) \leq \epsilon$ do
9      $J \leftarrow J \cup$ PPJ-R$(T_R, e_x.ptr, e_y.ptr, \epsilon, \theta)$;          // Recursion
10 return $J$;
```

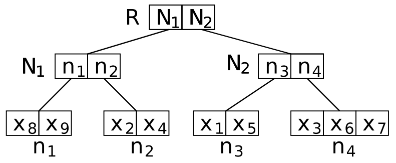


Figure 5: R-tree T_R for the collection R of Figure 2

Table 3: **Iterations performed by PPJ-R during the filtering phase for computing ST-SJOIN$(R, 0.2, 0.7)$ over collection R in Figure 2 (sf: suffix filter, ϵ: spatial filter)**

node join	object	checking pair	indexing
$n_1, n_2 \bowtie n_1, n_2$	x_2		$n_2.L_E = \{\langle x_2, 1\rangle\}$
	x_4		$n_2.L_A = \{\langle x_4, 1\rangle\}$
	x_8	$O_{x_8}[x_4] = 1$	$n_1.L_A = \{\langle x_8, 1\rangle\}$
	x_9	(x_9, x_4) pruned, ϵ	$n_1.L_A \bigcup= \{\langle x_9, 1\rangle\}$
		(x_9, x_8) pruned, **sf**	
$n_3, n_4 \bowtie n_3, n_4$			drop n_1, n_2 index
	x_1		$n_3.L_B = \{\langle x_1, 1\rangle\}$
	x_3		$n_4.L_D = \{\langle x_3, 1\rangle\}$
	x_5	(x_5, x_3) pruned, ϵ	$n_3.L_D = \{\langle x_5, 1\rangle\}$
	x_6	$O_{x_6}[x_5] = 1$	$n_4.L_C = \{\langle x_6, 1\rangle\}$
		$O_{x_6}[x_3] = 1$	
	x_7		$n_4.L_A = \{\langle x_7, 1\rangle\}$

EXAMPLE 5.4. *Figure 5 shows an exemplary R-tree T_R for the collection R of Figure 2 For* ST-SJOIN$(R, 0.2, 0.7)$*, the pair (N_1, N_2) is pruned because $dist_l(N_1, N_2) > 0.2$; thus,* PPJ-R *avoids checking all combinations of objects (x, y), with $x \in N_1$ and $y \in N_2$. Table 3 reports all the iterations performed by the algorithm during the filtering phase at level of leaf nodes.* PPJ-R *checks 6 pairs of objects similar to* PPJ-I *and* PPJ-C*.*

6. GROUPING OBJECTS

In this section, we propose a novel way of exploiting the prefix filtering principle that boosts the performance of both textual similarity join and ST-SJOIN. We demonstrate the intuition behind our technique with an example. Consider the collection of objects R in Figure 2 and, for now, a textual similarity join query over R w.r.t. a threshold $\theta = 0.7$. Table 4 shows $ppref(x)$ for every object x, using the $\ell_p^x = |x| - \lceil \theta \cdot |x| \rceil + 1$ formula. Observe that many of the objects share a common probe prefix. As a result, when examining for instance x_4, x_7, x_8 and x_9, any prefix-based algorithm, like [4, 10, 28], would probe postings lists L_A and L_B, and then, calculate and accumulate the same overlap O_x for all these objects. To make things worse, since $ipref(x) \subseteq ppref(x)$, the algorithm would index the same index prefix 4 times. On the other hand, assume

Table 4: Probe prefixes for the spatio-textual objects in Figure 2

object x	$x.text$	$ppref(x,), \theta = 0.7$
x_1	$\{B, C\}$	$\{B\}$
x_2	$\{E, F\}$	$\{E\}$
x_3	$\{D, E, F\}$	$\{D\}$
x_4	$\{A, B, E, F\}$	$\{A, B\}$
x_5	$\{C, D, E, F\}$	$\{C, D\}$
x_6	$\{C, D, E, F\}$	$\{C, D\}$
x_7	$\{A, B, C, D, F\}$	$\{A, B\}$
x_8	$\{A, B, D, E, F\}$	$\{A, B\}$
x_9	$\{A, B, C, D, E\}$	$\{A, B\}$

that we first group together all objects that have the same probe prefix, and next, we employ the same join algorithm over *groups* of objects *with the same prefixes*. This approach would clearly avoid the aforementioned overheads, and even more importantly would enable us to massively prune objects. For instance, consider the following two groups of objects, $\{x_5, x_6\}$ and $\{x_4, x_7, x_8, x_9\}$. Since the probe prefixes of these groups, $\{C, D\}$ and $\{A, B\}$, respectively, share no term, we can directly discard all possible pairs of objects by checking once the group objects. In contrast, PPJOIN would have to check 5 pairs: (x_5, x_4), (x_6, x_4), (x_7, x_5), (x_8, x_5) and (x_9, x_5).

On the other hand, extending PPJOIN to incorporate this grouping technique exhibits two shortcomings. First, it imposes an additional cost during the verification phase to unfold the objects for each pair of groups that satisfies the overlap threshold and identify the final results. Second, with respect to the actual objects of the collection, the grouping-based PPJOIN may change the order of examination. In other words, examining group objects by their size is not equivalent of examining the actual objects by their size, since objects of different lengths can have identical prefixes (e.g., x_4 and x_7 in our example). However, the correctness of PPJOIN relies on accessing the objects by increasing size. To tackle this problem we also consider the object sizes when defining the groups. Specifically, a group contains objects that (i) share a common probe prefix and (ii) have equal size. With this change, if for two groups $|g_x| > |g_y|$ holds then it is guaranteed that for every pair of objects $x \in g_x$ and $y \in g_y$, $|x| > |y|$ is also true, and therefore, PPJOIN can correctly use $ipref(g_x)$ to index a group object g_x.

The idea of grouping objects and then applying the join at the groups level can be employed by all our ST-SJOIN methods (i.e., PPJ, PPJ-I, PPJ-C and PPJ-R) to speed them up. In the following, we discuss the details of implementing it in each algorithm.

- PPJ considers exactly the same grouping of objects as PPJOIN does for a textual similarity join (i.e., there is one group for each distinct prefix and object size). In particular, before the execution of PPJ, the objects are ordered lexicographically and divided into groups based on their equal size and their common probe prefixes.
- PPJ-I/PPJ-C and PPJ-R are based on a spatial partitioning (i.e., grid-based or R-tree based). Grouping is applied independently at each spatial partition (cell or leaf node); therefore, two objects with the same prefix and size that belong to different spatial partitions are not grouped together. As a result, PPJ-I, PPJ-C and PPJ-R create a larger number of groups compared to PPJ. For example, in PPJ-I, the objects are ordered primarily by their cell id, secondarily by the size, and finally by the lexicographical order of their probe prefixes, and then they are divided into groups.
- To compute ST-SJOIN, PPJ, PPJ-I, PPJ-C and PPJ-R operate similarly to the case without grouping. Thus, PPJ and PPJ-I access the object groups by their size (length of their prefix) and PPJ-C and PPJ-R perform self and non-self joins between grid cells and leaf nodes respectively. Probing and indexing are performed at the groups level and thus, suffix filtering is of no use.
- Finally, during the verification phase, for each candidate pair of groups (g_x, g_y), all pairs of objects $(x, y) \in g_x \times g_y$ are verified by first checking whether $dist_l(x, y) \leq \epsilon$ and, then whether $sim_t(x, y) \geq \theta$.

Table 5: Parameters of synthetic data generation

parameter	**values**
$\|R\|$	30,000 100,000 **500,000** 1,000,000 3,000,000
$\|T\|$	5,000 10,000 **50,000** 100,000 500,000
SD	*uniform* or *clustered*
ϵ	0.001 0.005 **0.01** 0.05 0.1
θ	0.5 0.6 **0.7** 0.8 0.9

PPJ applies grouping without considering the spatial locations of objects; thus, it may create fewer groups and its filtering phase could be cheaper compared to that of PPJ-I/PPJ-C and PPJ-R. On the other hand, PPJ may bring in the same group objects that are very far from each other. For instance, PPJ groups x_7 together with x_8 and x_9 although x_7 is far from these objects. Thus, there may be many faraway object pairs verified by PPJ. In contrast, PPJ-I, PPJ-C and PPJ-R typically verify fewer object pairs than PPJ, as group pairs whose prefixes qualify the overlap threshold are immediately also checked against the spatial distance threshold, and may be pruned without having to unfold their contained objects.

7. EXPERIMENTAL EVALUATION

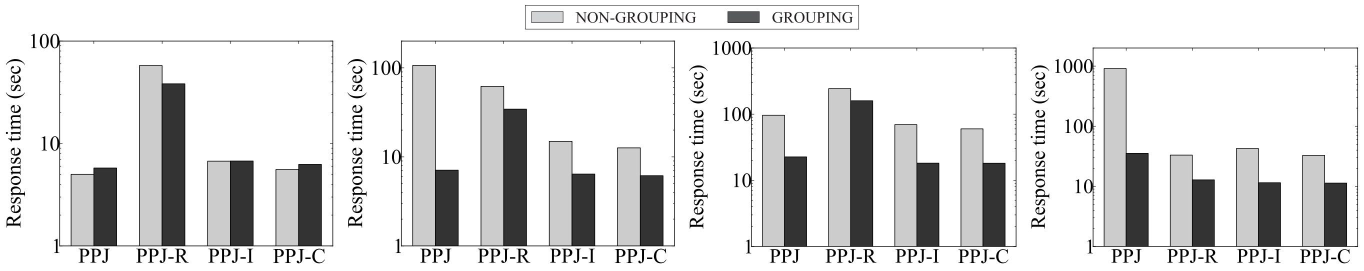
In this section, we present an experimental evaluation of our techniques for ST-SJOIN. Section 7.1 details the setup of our analysis. Section 7.2 investigates the contribution of the grouping technique. Section 7.3 demonstrates the need for approaches that efficiently consider both the spatial and the textual dimension of the problem, at the same time. Section 7.4 conducts an extensive performance analysis of our proposed methods. Finally, Section 7.5 investigates the potential of employing a spatio-textual index.

7.1 Setup

Our experimental analysis involves both real and synthetic spatio-textual collections. We use the following real data collections:

- FLICKR is a collection of photographs from Flickr for the city of New York taken over a period of 2 years. It contains 1,505,243 objects with a dictionary of 726,958 terms. For each photograph we used the union of its 'tags' and 'title' element of that image as its textual description. The weighted average size of an object is 10.5.
- POI-USCA and POI-AU are two collections of POIs and business listings for the state of California, USA and Australia, respectively, based on the SimpleGeo Places dataset[4]. For each place, we extracted its location and the union of its 'tag' and 'category', 'subcategory' elements as its textual description. POI-USCA contains 1,511,837 objects with a dictionary of 16,048 terms, while POI-AU contains 696,212 objects and 2,633 terms. The major difference between POI-USCA and POI-AU lies on the distribution of their objects in the two dimensional space. Due to the geography of Australia, the objects are extremely clustered, while the largest part of the space (i.e., the Australian desert) is empty. The weighted average size of an object is 4.4 for POI-USCA and 4.7 for POI-AU.

[4]https://simplegeo.com/products/places/



(a) FLICKR $\epsilon = 0.005, \theta = 0.9$ (b) POI-USCA $\epsilon = 0.005, \theta = 0.9$ (c) POI-AU $\epsilon = 0.05, \theta = 0.7$ (d) *clustered* default setup

Figure 6: Grouping versus Non-Grouping

We also generated synthetic datasets varying (i) the number $|R|$ of objects in a collection and (ii) the size $|T|$ of the global dictionary (see Table 5). To construct a collection, we fix one of the two parameters to its default value, shown in bold, and vary the other. Note that increasing $|T|$ enables us to study cases of datasets with dictionaries that are neither too large (e.g., as in FLICKR) nor too small (e.g., as in POI-USCA and POI-AU), compared to the number of objects $|R|$. In addition, for each combination of $|R|$ and $|T|$ values, we consider two possible spatial distributions of the objects (parameter SD): *uniform* or *clustered*. To construct a *uniform* collection, we generate locations uniformly in a $[0, 1]^2$ space. For a *clustered* collection, we first generate 10 uniformly distributed locations. Each of these locations acts as the center of a cluster. Then, for every new object x we randomly select one of the centers, p_c, and determine the position of x w.r.t. p_c, and thus, its location $x.loc$ in the space, following a Gaussian distribution with standard deviation $\sigma = 0.05$. Further, to capture the properties of real-world datasets, the textual description of an object is *correlated* to its spatial location. Thus, if two objects x and y are closely located in space then their textual descriptions share many terms. To achieve this, after generating the locations for every object of the collection, we consider a subset T_s of the global dictionary T containing $0.005 * |T|$ terms. Then, for each term $t \in T_s$, we randomly select from 1 to 3 objects, termed seeds and denoted by $S[t]$, and include t in their textual description. After that, we sort all objects by their minimum spatial distance to the seeds, and start defining their textual description. Every term $t \in T_s$ is included in the k nearest to its seeds objects, where k follows a zipfian distribution. The remaining $|T \backslash T_s|$ of the terms are assigned to objects randomly and their frequencies follow a zipfian distribution.

For the join methods evaluated, we measure their response times while varying the spatial distance threshold ϵ (relative to the side of the minimum square that encloses all data) and the textual similarity threshold θ. Table 5 shows the ranges of tested values for the synthetic collections; for the real data collections, we vary ϵ from 0.001 to 0.05 and θ from 0.6 to 0.9. Note that the setup $\epsilon = 0.001$ and $\theta = 0.9$ (tight thresholds) models a de-duplication scenario while looser values (e.g., $\epsilon = 0.05$ and $\theta = 0.7$) correspond to a recommendation or collaboration scenario. For `PPJ-R`, we consider a pre-constructed R-tree of page size 4KB, the construction time of which is not considered in the runtime of this method. On the other hand, the dynamic grid construction is included in the costs of `PPJ-I` and `PPJ-C`. Note that both the collections and the indexing structures used by the join methods are stored in main memory.

7.2 To Group or Not to Group

In the first experiment, we evaluate the effect of the grouping optimization (see Section 6) on the runtime of our join methods. For each of the `PPJ`, `PPJ-R`, `PPJ-I` and `PPJ-C` algorithms, we compare a grouping-based version against a non-grouping version, using FLICKR, POI-USCA, POI-AU and a *clustered* synthetic collection of spatio-textual objects. Figure 6 shows that the grouping-based version of each algorithm exhibits similar performance to the non-grouping version, in the worst case, but in the best case is significantly faster. Specifically, on POI-USCA, POI-AU and the synthetic collection, employing the grouping technique improves the performance by a few times to an order of magnitude. On the other hand, FLICKR does not favor grouping because for these data the great majority of groups have a single element; thus, the overhead of grouping balances its benefit. To justify this behaviour recall that the textual description of the objects in FLICKR is drawn from a large dictionary of approximately 700,000 terms. In practice, this means that, excluding the actual duplicates, the number of objects that share a common probe prefix is small, and therefore, the number of group records is close to the number of the initial objects contained in FLICKR. In summary, the use of grouping in all methods is at least as efficient as not using grouping, thus for the rest of our analysis, we always employ grouping in the tested methods.

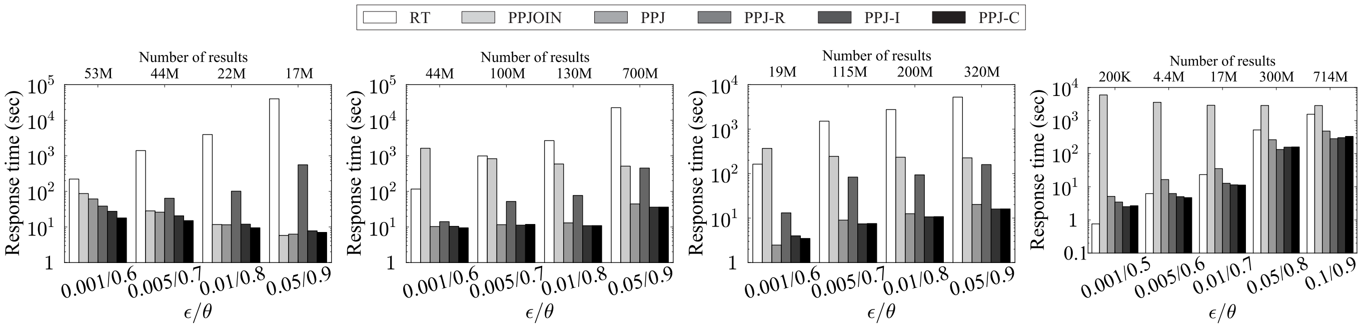
Finally, we also studied the effect of grouping on a plain textual similarity join. Table 6 reports the response time of the grouping-based and the non-grouping versions of `PPJOIN` for FLICKR and POI-USCA collections while varying the θ threshold. The results show that our grouping optimization can boost the performance of the state-of-the-art algorithm for textual similarity joins [28].

Table 6: Applying the grouping technique to `PPJOIN` for textual similarity joins: response time (sec)

	GROUPING		NON-GROUPING	
θ	FLICKR	POIS-USCA	FLICKR	POIS-USCA
0.6	67.24	570.97	72	1929.79
0.7	29.01	308.53	30.72	985.1
0.8	12.25	220.4	13.19	700.08
0.9	5.88	181.83	6.21	589.45

7.3 Comparison with Baseline Join Methods

We now compare our proposed ST-SJOIN methods against two baseline methods that deal separately with each dimension of the join: space and text. Specifically, the `RT` method indexes the objects of a collection using an R-tree. To evaluate ST-SJOIN, it performs a spatial ϵ-distance join and for each qualifying pair (x, y) it computes $sim_t(x, y)$ and verifies whether $sim_t(x, y) \geq \theta$. On the other hand, the `PPJOIN` method performs a textual similarity join as described in Algorithm 1 with the difference that during the verification phase it also checks whether $dist_l(x, y) \leq \epsilon$ holds for each candidate pair (x, y). Note that `PPJ` differs from `PPJOIN` in two ways: (i) `PPJ` carries out the $dist_l(x, y) \leq \epsilon$ check earlier, during



(a) FLICKR (b) POI-USCA (c) POI-AU (d) *clustered* (default setup)

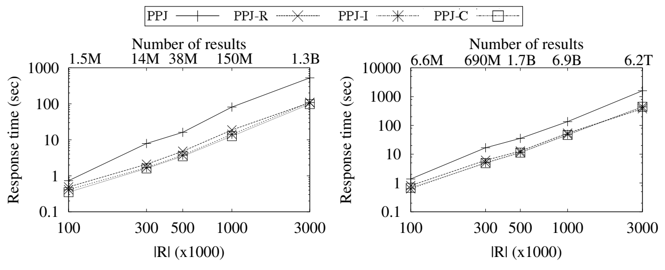
Figure 7: Comparison with RT and PPJOIN baseline methods

the filtering phase and (ii) like all of our ST-SJOIN methods, PPJ employs the grouping heuristic to improve search.

Figure 7 shows the response times of all methods for different values of ϵ and θ. As we vary the ϵ/θ ratio from $0.001/0.6$ to $0.05/0.9$ for the real collections (from $0.001/0.5$ to $0.1/0.9$ for synthetic data), the ST-SJOIN is progressively 'transformed' from a spatial ϵ-distance join (i.e., ϵ is tight, θ is loose) to a textual similarity join (i.e., ϵ is loose, θ is tight). As expected, the response time of the RT method increases while the time for PPJOIN decreases. With the exception of FLICKR, we also notice that the response time of the PPJ, PPJ-R, PPJ-I and PPJ-C algorithms also increases as they all consider the spatial dimension of the problem, in contrast to PPJOIN that takes into account ϵ only during the verification of pairs that pass the textual filter. To understand the case of FLICKR, recall that the collection contains a similar number of objects to POI-USCA but the dictionary of FLICKR is much larger than that of POI-USCA. This implies that, for FLICKR, ϵ is much less effective in pruning pairs compared to θ (if we put duplicates aside, the expected overlap between the contents of two objects is very low). An indicator for this issue is the difference in the trend of the join results as ϵ increases and θ decreases. Different to other datasets, for FLICKR, the number of results goes down due to θ becoming more tight, which means that for this collection ϵ has smaller effect in pruning compared to θ; i.e., ST-SJOIN is close to a textual similarity join, and therefore, PPJ, PPJ-I and PPJ-C exhibit similar behavior to PPJOIN.

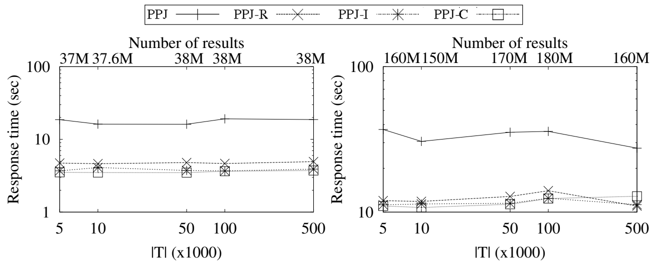
The poor performance of PPJ-R in all real datasets is attributed to the fact that due to the high spatial skew of these data, the R-tree employed by PPJ-R creates a large number of leaf nodes with much smaller extent compared to ϵ. This means that there is a huge number of leaf node pairs within distance ϵ, which have to be joined by PPJ-R (compared to the pairs of cells joined by PPJ-C). In addition, the leaf nodes contain much fewer objects compared to a cell of PPJ-I/PPJ-C, therefore the effect of grouping on PPJ-R is limited. We also experimented with R-trees of larger block size (e.g., 16K) and confirmed that the performance of PPJ-R improves in this case on the real data because leaf nodes become more populated and have larger extents. However, in other settings (e.g., synthetic data), PPJ-R becomes worse with the increase of the block size, as the extents of leaf nodes become much larger than ϵ in this case. On the other hand, PPJ-C always performs better than PPJ-R because its space partitioning is parametric to ϵ.

Finally, we observe that our advanced methods PPJ-R, PPJ-I and PPJ-C outperform both RT and PPJOIN in all cases, with the exception of $\epsilon = 0.001$ and $\theta = 0.5$ on the synthetic dataset. In this case, RT is superior because: (i) the spatial threshold ϵ is very tight,



(a) *uniform* (b) *clustered*

Figure 8: Synthetic collections: varying the number of objects $|R|$, with $|T| = 50{,}000$, $\epsilon = 0.01$, $\theta = 0.7$



(a) *uniform* (b) *clustered*

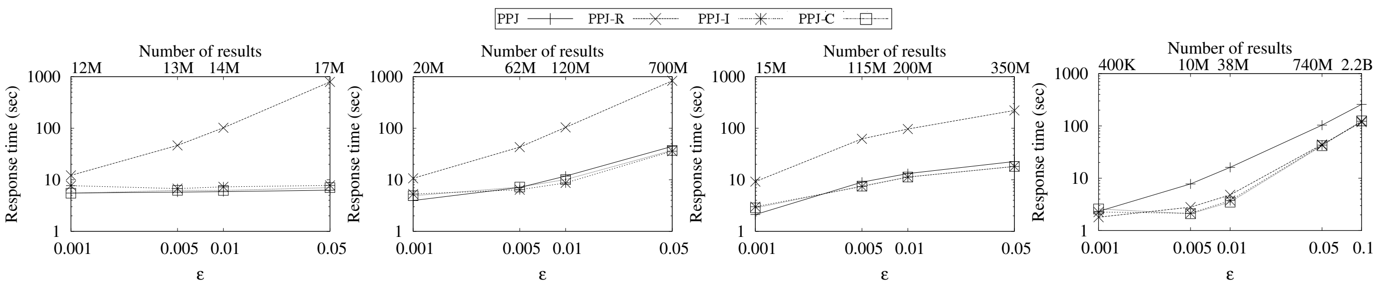
Figure 9: Synthetic collections: varying the size of dictionary $|T|$, with $|R| = 500{,}000$, $\epsilon = 0.01$, $\theta = 0.7$

and the spatial distance join is effective, as data are less clustered compared to the real datasets and (ii) θ is too loose for the textual filters used by our methods to have an effect; we observed that the number of pairs qualifying the spatial filter almost equals those that qualify both filters.

7.4 Comparison of ST-SJOIN methods

In the previous section, we demonstrated the superiority of our proposed methodology against baseline methods RT and PPJOIN. In this section, we conduct an extensive experimental analysis to identify the best technique among PPJ, PPJ-R, PPJ-I and PPJ-C.

Varying the number of objects $|R|$. Figure 8 illustrates the impact of varying the size of a synthetic collection R, on the ST-SJOIN computation. Naturally, when $|R|$ increases all methods required more time to compute the join. We note that PPJ-C exhibits the best



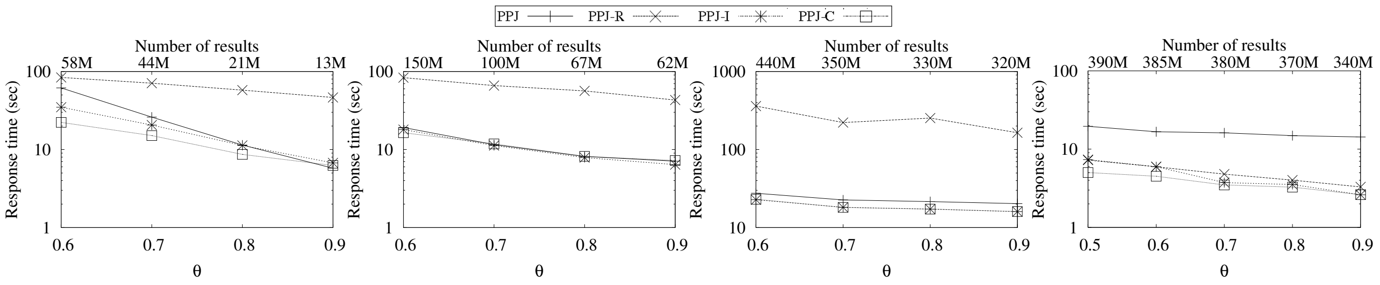
(a) FLICKR
$\theta = 0.9$

(b) POI-USCA
$\theta = 0.9$

(c) POI-AU
$\theta = 0.7$

(d) *uniform*
default setup

Figure 10: Varying the spatial distance threshold ϵ



(a) FLICKR
$\epsilon = 0.005$

(b) POI-USCA
$\epsilon = 0.005$

(c) POI-AU
$\epsilon = 0.05$

(d) *uniform*
default setup

Figure 11: Varying the textual similarity threshold θ

overall performance. On the other hand, as expected, the response time of PPJ is always higher than the time of the other methods since it does not employ any spatial indexing technique that would exploit ϵ to prune candidate pairs.

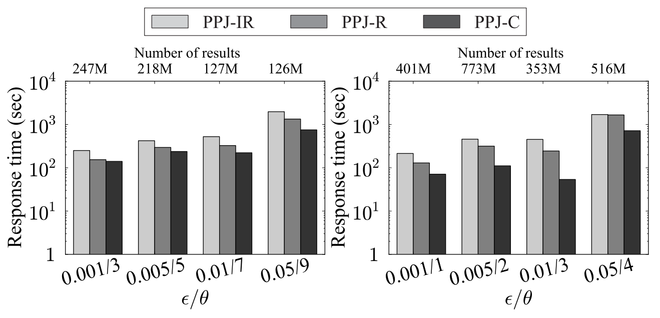
Varying the size of the dictionary $|T|$. Figure 9 shows that all methods are not affected significantly by the increase of $|T|$ in the synthetic data. The behavior of the methods is consistent with the number of the join results (almost stable). Finally, similarly to the case of varying $|R|$, PPJ-C has the best overall performance, and PPJ is the least efficient method.

Varying the spatial distance threshold ϵ. According to Figure 10, as ϵ increases, the join has more results and thus the response time of all methods also increases. However, we notice that in case of FLICKR, the time increase of PPJ, PPJ-I and PPJ-C is smaller compared to the other collections. As mentioned before, ST-SJOIN on FLICKR is very close to a textual similarity join; therefore, the number of results exhibit only a small increase, compared to the other cases, as the θ threshold is more important than ϵ. Thus, PPJ, PPJ-I and PPJ-C that extend a textual similarity method (PPJOIN) are not affected by ϵ. In contrast, PPJ-R slows down as ϵ increases. Recall, from Section 7.3, that the extents of the R-tree leaf nodes are very small and they are spatially skewed, therefore the pairs of leaf nodes joined in PPJ-R increases significantly with ϵ. PPJ-C is, in general, the best method, with either PPJ or PPJ-I having comparable response times in some setups. An interesting observation is that while the less efficient method on the real collections is PPJ-R, PPJ performs the worst on the synthetic data. The reason behind this is that the synthetic data are less clustered compared to the real data and, in addition, the spatial autocorrelation is also lower in this case. In other words, the join is less close to a textual similarity join in this case (this is also consistent with the results of Figure 7). Thus, PPJ-R performs similarly to PPJ-C, as these two methods define similar spatial partitions, while PPJ fails to exploit ϵ to prune group pairs early. Finally, we also observe that for the synthetic collection the response time of PPJ-I and PPJ-C drops as ϵ is increased from 0.001 to 0.005, although the join contains more results. This is because the cost of performing the dynamic grid partitioning drops with ϵ and it constitutes an important factor in this case, where the overall join cost is very low.

Varying the textual similarity threshold θ. Figure 11 shows that when θ increases the response times of all methods decrease due to the reduction of the join result. The phenomenon is less intense in case of POI-USCA and POI-AU, where the decrease of the join results is very small. Similar to the previous experiments, PPJ-C is the most efficient method and PPJ-R and PPJ are the least efficient methods for the real and the synthetic collections, respectively.

7.5 The IR-tree and the PPJ-IR Method

Finally, we investigate whether the recently proposed spatio-textual IR-tree index [12] can facilitate the computation of a ST-SJOIN (see Section 2.3 for a detailed description of this index). If the data are indexed by an IR-tree, we can extend PPJ-R to a PPJ-IR algorithm, as follows. For the currently examined pair of non-leaf nodes (N_x, N_y), we merge the inverted files linked to them to compute for each pair of entries (e_x, e_y) in $N_x \times N_y$ an upper bound of the textual overlap between any pair of objects (x, y) indexed under the corresponding pair of nodes $(e_x.ptr, e_y.ptr)$. This bound is used together with the spatial distance lower bound $dist_l(e_x, e_y)$ to prune pair (e_x, e_y) if either of these bounds does not meet the respective join thresholds θ and ϵ. Therefore, PPJ-IR uses a textual filter for non-leaf entries which cannot be employed by PPJ-R. Still, PPJ-IR has the overhead of accessing and merging the inverted files



(a) FLICKR (b) POI-USCA

Figure 12: Employing the IR-tree, comparison for ST-SJOIN

of non-leaf nodes and, in addition, the overlap bound cannot be converted to a tight Jaccard similarity bound (as the sizes of x and y can be large). In order to favor PPJ-IR, in our experiment we use overlap instead of Jaccard similarity and compare PPJ-IR to PPJ-R and PPJ-C on FLICKR and POI-USCA while varying the ratio ϵ/θ of the thresholds (Figure 12). For POI-USCA we consider smaller θ values compared to FLICKR, as the weighted average object size is lower. Note that PPJ-IR fails to outperform PPJ-R, although it may, in some cases, consider fewer entry pairs than PPJ-IR, due to the use of the additional overlap filter. This happens because the cost of processing a pair of internal nodes N_x and N_y on PPJ-IR is much higher compared to PPJ-R, due to the accessing and merging of the (potentially large) inverted files attached to the nodes. We also experimented with the DIR-tree [12], a variant of the IR-tree, but it performed even worse than the IR-tree.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we identified and studied the spatio-textual similarity join (ST-SJOIN), a query with a wide range of applications (social recommendation, near-duplicate detection, etc.). We proposed methods that combine ideas from state-of-the-art spatial and textual join approaches; our best method (PPJ-C) dynamically partitions the space according to the spatial predicate of the join and then confines the application of the textual predicate only to pairs of adjacent grid cells. We proposed a grouping optimization technique which boosts the performance of all textual and spatio-textual join methods. Finally, we conducted extensive experiments on real and synthetic datasets to evaluate the performance of our methods.

Although our study was limited to specific measures for textual similarity (i.e., Jaccard similarity and overlap), our methods can easily be adapted to be used with other measures (see [28] on how PPJOIN can be adapted for a variety of measures). In addition, besides the fact that PPJ-C is the most efficient method, we note that it can be easily applied in a parallel processing environment. For example, after dynamic grid partitioning, each cell c is assigned to a computer node v and v is given a copy of the objects contained in c and all cells in $A[c]$; PPJ is then run at each node and there is no need for communication among the nodes, since join results are independent and there are no duplicates. In the future, we plan to investigate the potential of such an implementation. In addition, we plan to study the application and evaluation of alternative types of spatio-textual joins (e.g., a set-containment join enriched with spatial distance/containment constraints).

9. REFERENCES

[1] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *SIGIR*, 2004.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, 2006.

[3] J. Ballesteros, A. Cary, and N. Rishe. Spsjoin: parallel spatial similarity joins. In *GIS*, pages 481–484, 2011.

[4] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.

[5] C. Böhm, B. Braunmüller, F. Krebs, and H.-P. Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *SIGMOD Conference*, 2001.

[6] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD Conference*, 1993.

[7] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1):373–384, 2010.

[8] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD Conference*, 2011.

[9] E. P. F. Chan. Buffer queries. *IEEE Trans. Knowl. Data Eng.*, 15(4):895–910, 2003.

[10] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.

[11] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD Conference*, 2006.

[12] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[13] J. Ding, L. Gravano, and N. Shivakumar. Computing geographical scopes of web resources. In *VLDB*, 2000.

[14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[15] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, 2008.

[16] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.

[17] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, 1984.

[18] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSDBM*, 2007.

[19] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[20] 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.*, 23(4):585–599, 2011.

[21] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging with local lexicons to build indexes for textually-specified spatial data. In *ICDE*, 2010.

[22] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *SIGMOD*, 1986.

[23] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *SSTD*, 2001.

[24] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, 2011.

[25] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD Conference*, 2004.

[26] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *SSTD*, 2005.

[27] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *ICDE*, 2009.

[28] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15, 2011.

[29] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, 2009.

[30] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, 2005.

[31] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.*, 23(4):453–490, 1998.