Similarity Search on Spatio-Textual Point Sets

Christodoulos Efstathiades
European University Cyprus
Cyprus
C.Efstathiades@euc.ac.cy

Alexandros Belesiotis
IMIS, R.C. Athena
Greece
abelesiotis@imis.athena-innovation.gr

Dimitrios Skoutas
IMIS, R.C. Athena
Greece
dskoutas@imis.athena-innovation.gr

Dieter Pfoser
George Mason University
USA
dpfoser@gmu.edu

ABSTRACT

User-generated content on the Web increasingly has a geospatial dimension, opening new opportunities and challenges in location-based services and location-based social networks for mining and analyzing user behaviors and patterns. The applications of such analysis range from recommendation systems to geo-marketing. Motivated by these needs, querying and analyzing spatio-textual data has received a lot of attention over the last years. In this paper, we address the problem of matching point sets based on the spatio-textual objects they contain. This is highly relevant for users associated with geolocated photos and tweets. We formally define this problem as a Spatio-Textual Point-Set Join query, and we introduce its top-k variant. For the efficient treatment of such queries, we extend state-of-the-art algorithms for spatio-textual joins of individual points to the case of point sets. Finally, we extensively evaluate the proposed methods using large scale, real-world datasets from Flickr and Twitter.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms

Keywords

spatio-textual join, spatio-textual point sets, similarity search

1. INTRODUCTION

Social media platforms such as Twitter, Flickr, Facebook and Foursquare have attracted billions of active users. In the case of Twitter 500 million tweets are exchanged every day from 100 million active users. User activities in these platforms generate content that has *textual component*, e.g., status updates, short messages, or tags, and, following the widespread adoption of GPS in mobile devices, a *geospatial component*, e.g., geotagged tweets, photos, and user check-ins. Thus, the actions of users are documented by their messages in social networks and as such generate "traces", which consist of spatio-textual objects.

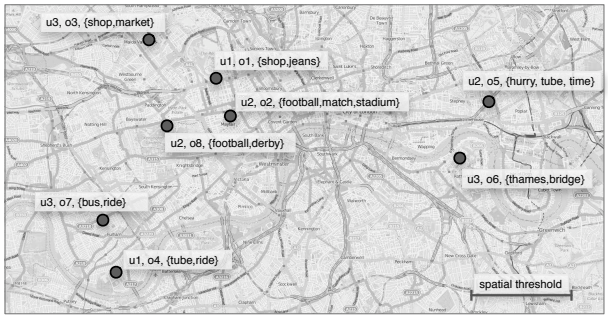


Figure 1: `STPSJoin` query scenario. Multiple objects are spatially or textually similar, but only users u_1 and u_3 have objects which are mutually similar.

Efficient indexing and querying of spatio-textual data has received a lot of attention over the past years, due to the high importance of such content in location-based services, such as nearby search and recommendations. In particular, multiple types of spatio-textual queries have been extensively studied, including boolean range queries, top-k queries, k-nearest neighbor queries, and more recently, spatio-textual similarity joins [11, 7]. Nevertheless, in existing works, *spatio-textual entities are typically treated as isolated observations*. A typical example query is to find nearby restaurants or hotels matching certain criteria.

The work in [7] deals with finding pairs of entities that are both spatially close and textually similar. Example use cases are de-duplicating Points-of-Interest across datasets, or finding matching photos taken at roughly the same location and having similar tags.

Now consider looking for similar users in social networks. Here, a user is characterized by the messages they generate and, if available, respective location information. As such, each message can be considered a spatio-textual object, e.g., a geotagged photo or tweet. With each user being character-

ized by a set of spatio-textual objects, to find similar users, one needs to examine the similarity of these respective sets. Effectively, this characterizes users by what and where they tweet. An example of such a scenario is depicted in Figure 1. To that effect, this work addresses the problem of similarity search for *spatio-textual entities*, with an entity being characterized by a *set of spatio-textual objects*. We introduce the *Spatio-Textual Point-Set Similarity Join* (`STPSJoin`) query. Given sets of spatio-textual objects, each one belonging to a specific entity, this query seeks pairs of entities that have similar spatio-textual objects.

The `STPSJoin` can naturally model the search for entities exhibiting similar behavior according to the spatio-textual objects they generate. With social media users posting messages at various locations, the `STPSJoin` allows us to discover users that exhibit similar "geo-textual" behavior. This holds especially true for location-based social media sites, e.g., Foursquare, where users report on the places they visit. Following the general observation that spatio-textual object sets can be used to define the context of a user, e.g. has family because of frequent toy store visits, the `STPSJoin` can be used to discover such groups of similar users.

To efficiently process (top-k) `STPSJoin`, we adapt and extend the state-of-the-art algorithms for processing similarity joins for single points [7]. The proposed algorithms make use of spatio-textual indexes in conjunction with an early termination and a filter-and-refinement strategy to effectively prune the search space, thus reducing the execution time by orders of magnitude. More specifically, the contributions of our work are as follows.

- We formally define the spatio-textual point set similarity join (`STPSJoin`) query, which extends and generalizes the spatio-textual similarity join for the case of point sets, and its top-k variant.
- We first derive a baseline algorithm for the `STPSJoin` query by adapting the state-of-the-art `PPJ-C` algorithm [7] to work for point sets. Then, we propose two optimized algorithms, `S-PPJ-B` and `S-PPJ-F`, which apply an early termination and a filter-and-refinement strategy, respectively, to drastically prune the search space. This significantly reduces the number of comparisons required, both in terms of pairs of entities and in terms of individual points for each candidate pair.
- In addition, we present an alternative version of `S-PPJ-F`, denoted as `S-PPJ-D`, which relies on an R-tree instead of a grid for the spatial indexing.
- We adapt our methods to efficiently treat the top-k `STPSJoin` query. We provide a direct adaptation of our best performing algorithm, and extended it in order to allow additional pruning of the search space.
- Finally, we perform an extensive experimental evaluation using three large, real-world datasets. The results of the experimental evaluation demonstrate that the proposed algorithms achieve an order of magnitude and above improvement in terms of execution time when compared to the baseline method.

The remainder of this work is structured as follows. Section 2 reviews related work. The `STPSJoin` query and its top-k variant are formally introduced in Section 3. The algorithms for the efficient evaluation of (top-k) `STPSJoin` queries are presented in Section 4. Section 5 presents an experimental evaluation of the proposed approaches. Finally, Section 6 gives conclusions and directions for future work.

2. RELATED WORK

First, we review recent advances on spatio-textual search, which exploit spatial and textual characteristics in order to efficiently prune the search space while searching for similar objects. Then, we detail the state-of-the-art in similarity joins, in order to establish the basis for our work on point set joins. Finally, we present related literature on user recommendations using location histories.

2.1 Spatio-Textual Search

A large amount of web documents nowadays contain both spatial and textual information, characteristics which are exploited by modern applications to provide enhanced location-based services. Such applications rely on spatio-textual indexing for efficient computation.

Spatio-textual indexes. Current research enables the combination of spatial and textual indexes into hybrid spatio-textual indexes that explicitly support geographically aware search. Established spatial indexes, such as R-trees [22], regular grids, and space-filling curves, are integrated with textual indexes, such as inverted files or signature files. For example, SPIRIT [36] uses regular grids as spatial indexes and inverted files for the indexing of the documents. [43] proposed different approaches to hybrid indexing that employ R*-trees [6] for spatial indexing and inverted files for textual indexing. [12] follows a similar approach, but utilizes space-filling curves for spatial indexing. [15] combines R-trees with signature files, which are stored internally in the nodes of the tree. The IR-tree index [13, 26], leverages inverted files for each node of the tree, in order to keep aggregate information of the textual characteristics of the relevant objects, while [33] uses aR-Trees [30] in combination with inverted files. Spatio-textual search is employed in order to answer a range of queries. An overview of the performance of the most commonly used spatio-textual indexes in such queries can be found in [11].

Spatial group keyword queries. Another type of spatio-textual queries is spatial group keyword queries [40, 41, 9, 29]. These aim at finding groups of spatio-textual objects that collectively satisfy a number of given keywords, while minimizing the collective distances between points in the group and the given query point.

Although the aforementioned queries involve searching for groups of objects, they differ from the problem addressed in this paper. The `STPSJoin` query is not constrained by an input query point nor a given textual description. In addition, groups of objects are predefined according to the user they are associated with. `STPSJoin` considers spatial and textual distances of objects across groups, rather than within groups. Finally, `STPSJoin` deals with the problem of spatio-textual join, which is fundamentally different from range and kNN queries.

2.2 Similarity Joins

Similarity joins seek to identify pairs of objects from given sets that satisfy a predefined similarity threshold.

Set similarity joins. The set similarity join task is computationally challenging; a naive approach requires the consideration of the similarity between every possible pair of objects across sets. Set similarity joins have been extensively studied, especially with respect to textual characteristics, and multiple optimizations have been proposed. [34] uses an inverted index based probing method to reduce the number of potential candidates. [10] observes that the prefixes of potential candidates must satisfy a minimal overlap. The `ALL-PAIRS` algorithm proposed by [5] further optimizes the size of the inverted index. [37] presents `Adapt-Join` and [38] proposes `PPJOIN+` which are the state-of-the-art algorithms for set similarity joins. `PPJOIN+` builds on `ALL-PAIRS` and introduces a *positional filtering principle* which exploits the ordering of tokens, and operates both on the prefix and the suffix of the tokens of objects. `PPJOIN+` is internally used as the final step of our algorithms in order to efficiently compute textual similarity joins. An experimental analysis and evaluation on string similarity joins can be found in [24].

Spatial joins. Data structures and algorithms for spatial joins have been widely studied in the literature. A relevant survey can be found in [23]. Spatial joins have been used in combination both with space partitioning as well as with data partitioning structures. The state of the art algorithm for spatial joins has been proposed in [8]. We utilize this algorithm to prune the search space when searching for spatially relevant users using an R-Tree (see Section 4.1.4).

The problem of spatial joins over point sets has not received much attention. Adelfio et al. [2, 1] focus on similarity search for a collection of spatial point set objects based on the Hausdorff distance. The motivation behind their work is highly relevant to the `STPSJoin` query. However, there are important differences. We consider web objects with spatio-textual characteristics and measure the distance among point sets using a different similarity measure. The Hausdorff distance measures the maximum discrepancy between two point sets, whereas in our work we use a measure inspired by the Jaccard coefficient which focuses on the amount of objects from different point sets that are similar.

Spatio-textual joins. Spatio-textual joins have attracted some attention recently with a specific focus on joins for spatio-textual points. This process is primarily executed for the purpose of duplicate detection. The work in [3] is one of the first examples of spatio-textual join methods. They propose the SpSJoin query that follows the MapReduce paradigm for scalable computation of spatio-textual join queries. The spatio-textual join query has been also studied in the form of spatial regions associated with textual descriptions ([27, 28, 20]). Pruning strategies, based on spatial and textual signatures of objects, are employed to filter the number of candidates. [32] presents grid and quad tree based indexes in order to efficiently partition the database either in a local or global fashion. They also explore different dimensions of the problem, including the use of `PPJOIN+` and `All-Pairs` for text similarity joins, as well as single and multi-threaded approaches.

Bouros et al. [7] propose the state of the art spatio-textual join algorithms. Their work builds on top of `PPJ`, a baseline method that extends `PPJOIN+` to account for objects with spatio-textual characteristics and a given spatial distance threshold. The algorithms `PPJ-C` and `PPJ-R` extend `PPJ` by leveraging a grid and an R-Tree based index respectively. These methods provide the basis for our work; thus, we revisit them in more detail in Section 4.1.1.

Work on spatio-textual joins is highly relevant to our approach. However, the focus is different. To the best of our knowledge, current research in the field has focused on spatio-textual similarity joins among points. On the contrary, our work introduces spatio-textual similarity joins among point sets. Point sets are relevant when objects are grouped with respect to a common characteristic. In this case, the focus is on identifying similarities among groups, rather than single elements. For instance, in the case of web objects, a group consists of objects associated with the same user. In this case, point-set joins identify user similarity instead of object similarity.

2.3 User Recommendation Systems

Matching users based on their location history is one of the main tasks of recommendation engines in location-based social networks [4]. User location histories have been used for identifying local experts, recommending friends, and extracting local communities. It has been revealed by several studies that location information plays a vital role in determining such relationships [14, 16].

Typically, these approaches take into consideration additional information, such as location ratings, semantics of location descriptions and tags, sequence of visit or duration of stay. [25] identifies users with similar traveling patterns based on matching sequences of locations visited by the users. Similar works ([25, 42, 39]) deal with finding users with similar patterns in behavior. [21] study several features to identify users that are similar to a given user. Their methods are based on a logistic regression model. According to their work, a single and in many cases imprecise user location feature (such as city or country) is not effective.

In this paper, we focus on multiple geo-tagged objects for each user, which may provide a better insight into location-based user similarity.

3. PROBLEM DEFINITION

We assume a database $\mathcal{D}$ of spatio-textual objects created by different users U. A spatio-textual object $o \in \mathcal{D}$ is a triple $o = \langle u, loc, doc \rangle$, where $u \in U$ is the user associated with this object, $loc = \langle x, y \rangle$ is a spatial point and doc is a set of keywords. We refer to the user, location and keywords associated with an object o using the notation $o.u$, $o.loc$ and $o.doc$ respectively. In addition, we use D_u to denote the set of objects belonging to user u.

The spatial distance $\delta(o, o')$ between two objects is calculated as the Euclidean distance between their spatial locations. Moreover, the textual similarity $\tau(o, o')$ is measured according to the Jaccard similarity of their keywords:

$$\tau(o, o') = \frac{|o.doc \cap o'.doc|}{|o.doc \cup o'.doc|}.$$

Given a spatial threshold ϵ_{loc} and a textual threshold ϵ_{doc}, we say that two objects $o, o' \in \mathcal{D}$ match if their spatial distance is below ϵ_{loc} and their textual similarity is above ϵ_{doc}. Matching between objects is defined using the predicate μ:

$$\mu(o, o') = \begin{cases} True & \text{if } \delta(o, o') \leq \epsilon_{loc} \text{ and } \tau(o, o') \geq \epsilon_{doc} \\ False & \text{otherwise.} \end{cases}$$

For brevity, we overload μ to account for matching an object

with a set of objects $D \subseteq \mathcal{D}$:

$$\mu(o, D) = \begin{cases} \textit{True} & \textit{if there exists } o' \in D \textit{ such that } \mu(o, o') \\ \textit{False} & \textit{otherwise.} \end{cases}$$

Furthermore, let two spatio-textual point-sets D and D'. Function $M(D, D')$ returns the set of objects in D that match with at least one object in D':

$$M(D, D') = \{o \in D \textit{ such that } \mu(o, D')\}\,.$$

We then use M to define the similarity of point-sets D and D'. In particular, this is measured as the fraction of the matched points from one set to the other divided by the total number of points in the two sets. Formally:

$$\sigma(D, D') = \frac{|M(D, D')| + |M(D', D)|}{|D| + |D'|}\,.$$

The employed measure is inspired by the Jaccard similarity, which is not directly applicable since it does not support partial similarity between elements. More elaborate similarity metrics over point sets can be found in [19, 31].

We can now define the *Spatio-Textual Point Set Join* query (`STPSJoin`). `STPSJoin` identifies all pairs of users U which are associated with sets of spatio-textual objects that have a match higher than a specified threshold ϵ_u. We assume a total ordering over U (i.e. $\prec_U$) to avoid returning duplicate pairs. Formally, the `STPSJoin` query is defined as follows.

Definition 1. *Given a database $\mathcal{D}$ of spatio-textual objects belonging to a set of users U, the `STPSJoin` query is a tuple $Q = \langle \epsilon_{loc}, \epsilon_{doc}, \epsilon_u \rangle$ which returns a set R containing all pairs of users (u, u') such that $u, u' \in U$, $u \prec u'$, and $\sigma(D_u, D_{u'}) \geq \epsilon_u$ with respect to the spatial and textual thresholds ϵ_{loc} and ϵ_{doc}.*

An extension of the `STPSJoin` query in which we seek only the k best pairs of users, in terms of spatial and textual similarity of their objects, is the top-k `STPSJoin` query. Formally, the top-k `STPSJoin` query is defined as follows.

Definition 2. *Given a database $\mathcal{D}$ of spatio-textual objects belonging to a set of users U, the top-k `STPSJoin` query is a tuple $Q = \langle \epsilon_{loc}, \epsilon_{doc}, k \rangle$ which returns a set R containing k pairs of users (u, u') such that $u, u' \in U$, $u \prec u'$, and for any pair of users $(v, v') \notin R$ it holds that $\sigma(D_u, D_{u'}) \geq \sigma(D_v, D_{v'})$ for each $(u, u') \in R$ with respect to the spatial and textual thresholds ϵ_{loc} and ϵ_{doc}.*

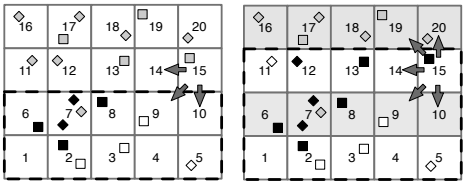
4. ALGORITHMS

This section presents algorithms for the evaluation of the `STPSJoin` query and the top-k `STPSJoin` query. First, we present a baseline algorithm, and then we introduce methods that exploit a filter and refine strategy in combination with spatio-textual indexes in order to direct the search. Then, we explain how our methods can be adapted to account for the top-k `STPSJoin` query.

4.1 Algorithms for STPSJoin

4.1.1 Baseline Approach

Preliminaries. The straightforward method for evaluating an `STPSJoin` query is to find, for every pair of users, the set of matching objects, and then to check whether the resulting similarity score σ exceeds the specified threshold ϵ_u. Thus, for a pair of users (u, u'), the problem can be cast as a spatio-textual similarity join query, ST-SJOIN$(D, \epsilon_{loc}, \epsilon_{doc})$, which has been studied in [7]. This query returns all pairs of objects (o, o') in D such that $o, o' \in D$, $\delta(o, o') \leq \epsilon_{loc}$ and $\tau(o, o') \geq \epsilon_{doc}$. Based on this, we can find the objects of u that match with those of u', and vice versa, and then proceed with computing the score σ for this pair of users.



(a) *PPJ-C* traversal. (b) *PPJ-B* traversal.

Figure 2: *PPJ-C* and *PPJ-B* grid traversal strategies examples. Objects associated with user u (u') are depicted by squares (diamonds). Matched objects are painted black, objects that do not match are painted white, while objects whose state has not been determined are painted grey. *PPJ-B* has determined the state of every object in cells 1 to 15, while *PPJ-C* only the objects until cell 10.

For this purpose, we adapt the PPJ-C algorithm from [7] for the purposes of ST-SJOINs. PPJ-C uses a grid to partition the space, in order to limit the search to those candidates that can satisfy the spatial predicate of the join. The grid is constructed dynamically at query time, using cells that have an extent in each dimension that equals the spatial distance threshold ϵ_{loc}. The cells are assigned ids in a row-wise order from bottom to top (see Figure 2a).

PPJ-C visits the cells in ascending order of their ids, taking advantage of the spatial filtering, since the objects in each visited cell c need to be joined only with those in c and in the cells adjacent to c. In fact, to avoid duplicates, only the adjacent cells with ids lower than c need to be examined. Thus, for each cell, one self-join operation and at most four non-self join operations need to be performed. These are performed using the PPJ algorithm, that in turn extends the set similarity join algorithm PPJOIN [38] by including an additional check on the spatial distance of two objects.

The `S-PPJ-C` algorithm. Using `PPJ-C` as basis, we can derive a baseline algorithm, denoted as `S-PPJ-C` (Set-PPJ-C), for the `STPSJoin` query. `S-PPJ-C` is presented in Algorithm 1. During the construction of the grid, we maintain the following additional information: (a) for each cell c, we maintain the contained objects in separate lists according to the user they belong to; we denote by D_c^u the set of objects of user u that are contained in c; (b) for every user u, we maintain a list of cells C_u that contain objects belonging to u; C_u is sorted according to cell ids in ascending order.

The `S-PPJ-C` algorithm loops through all pairs of users, taking into consideration the total ordering $\prec_U$ of the user set U. For each pair of users (u, u'), `S-PPJ-C` executes a non-self join version of the `PPJ-C` algorithm from [7] presented above. The difference with the standard `PPJ-C`, lies in the fact that in this case pairs with objects from both users are returned. To do so, first the lists C_u and $C_{u'}$ containing the cells for u and u' respectively are gathered. Next, the algorithm iteratively selects from either list the cell c with the lowest id that has not been selected yet. Assume that

Algorithm 1: S-PPJ-C Algorithm

Input: D, U, ϵ_{doc}, ϵ_{loc}, ϵ_u
Output: Pairs of matched users R

1 $R \leftarrow \emptyset$
2 $selectedUsers \leftarrow \emptyset$
3 $G \leftarrow createGridIndex(D, U, \epsilon_{loc})$
4 **foreach** $u_1 \in U$ **do**
5 **foreach** $u_2 \in selectedUsers$ **do**
6 $r \leftarrow PPJ\text{-}C(u_1, u_2, \epsilon_{doc}, \epsilon_{loc})$
7 $\sigma \leftarrow \frac{|r|}{|D_{u_1}|+|D_{u_2}|}$
8 **if** $\sigma \geq \epsilon_u$ **then**
9 $R.add(\langle u_2, u_1 \rangle)$
10 $selectedUsers.add(u_1)$
11 **return** R

the next selected cell c is from the list of user u. For every cell c' in $C_{u'}$ with $c'.id \geq c.id$, a non-self join version of PPJ is executed with input the spatio-textual point sets D_u^c and $D_{u'}^{c'}$. Since C_u and $C_{u'}$ may both contain the cell c, we avoid the duplicate execution of PPJ for c.

The results of PPJ-C are used to compute the user similarity score σ (line 6-7). Pairs of users that achieve a similarity score above the threshold ϵ_u are collected in the result set.

4.1.2 The S-PPJ-B Algorithm

The drawback of the S-PPJ-C algorithm is that for each pair of users it finds all their matching points and computes the exact value of their similarity score σ before checking whether this exceeds the given threshold. Instead, since we are only interested in finding those pairs with a similarity that exceeds ϵ_u, we can reduce the execution time of the algorithm by terminating the computation for a pair of users as soon as it can be decided that their similarity is below ϵ_u. Following this observation, we derive a more efficient algorithm, denoted as S-PPJ-B (where B stands for bound).

S-PPJ-B operates in the same manner as S-PPJ-C, with the only difference that it replaces the execution of PPJ-C with a modified process, denoted as PPJ-B. PPJ-B leverages the use of an upper bound on the number of unmatched objects for a pair of users to effectively prune the search on the spatial grid. More specifically, the intuition behind PPJ-B is the following. While examining two users, PPJ-B leverages the user similarity threshold ϵ_u and the number of objects belonging to each user in order to compute an upper bound on the number of unmatched objects between the two users, above which the user similarity cannot exceed ϵ_u. In the following, we first derive this upper bound, and then we explain the process followed by PPJ-B in order to allow for early termination during the examination of two users.

For a pair of users (u, u'), let $\beta_{u,u'}$ denote the number of objects from user u and user u' that do not match with the other user, i.e.:

$$\beta_{u,u'} = |D_u| + |D_{u'}| - |M(D_u, D_{u'})| - |M(D_{u'}, D_u)|$$

An upper bound for $\beta_{u,u'}$ is derived as follows.

LEMMA 1. *For a pair of users (u, u'), if $\beta_{u,u'} > (1-\epsilon_u) \cdot (|D_u| + |D_{u'}|)$ then $\sigma(D_u, D_{u'}) < \epsilon_u$.*

PROOF. The proof is derived from the definition of the similarity score between two users, as follows:

$$\sigma(D_u, D_{u'}) \geq \epsilon_u \Rightarrow \frac{|M(D_u, D_{u'})| + |M(D_{u'}, D_u)|}{|D_u| + |D_{u'}|} \geq \epsilon_u \Rightarrow$$
$$\frac{|D_u| + |D_{u'}| - \beta_{u,u'}}{|D_u| + |D_{u'}|} \geq \epsilon_u \Rightarrow 1 - \frac{\beta_{u,u'}}{|D_u| + |D_{u'}|} \geq \epsilon_u \Rightarrow$$
$$\beta_{u,u'} \leq (1-\epsilon_u) \cdot (|D_u| + |D_{u'}|)$$

□

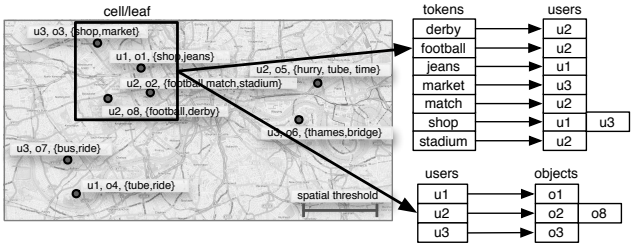


Figure 3: Spatio-textual structure for S-PPJ-F and S-PPJ-D.

Upon traversing a cell c, PPJ-C checks for potential matches in cells with ids lower than $c.id$. Therefore, we cannot be certain that objects that have not been matched so far will also not match with objects in cells with higher ids (i.e. in the next cell or row). Therefore, the bound may be used within PPJ-C, but only with respect to the objects discovered from the beginning of the grid until the previous row. The objects that were traversed in the current row have to be excluded from calculation.

To that end, PPJ-B devises a different grid traversal strategy that allows the pruning mechanism to utilize every object appearing in cells traversed when the bound evaluation is executed. Specifically, this strategy traverses the rows from bottom to top (considering the id of the bottom row as 1), and depending on whether the id of a row is odd or even, different treatment is followed. If a cell $c_{i,j}$ belongs to a row with odd id, i.e. j is odd, then the objects contained in it are matched with objects from all surrounding cells, except the cell directly on the right, i.e. $c_{i+1,j}$. Matching is done by executing PPJoin. Otherwise, if the cell belongs to an even row, then we match its objects only with objects from the other user from the cell that is directly on the left, i.e. $c_{i-1,j}$. This process is illustrated in Figure 2b.

Following this traversal strategy, PPJ-B allows for early termination using the bound β, while still maintaining the property of PPJ-C to avoid duplicate examination of the same pair of cells. Indeed, when PPJ-B traverses the last cell of an odd row, it has considered every potential match for any object it has encountered up to that point. Thus, it checks whether the number of objects that have not been matched exceeds the calculated bound β. If so, the search stops, since it is impossible to result in a user similarity score that exceeds ϵ_u. Note that, in practice, since the grid may be rather sparse, some rows may be empty. In that case, when the next visited cell belongs to a row that is not directly above the previous one, the same check can be performed, even if the last examined row was even, since previously encountered objects cannot have any future matches.

4.1.3 The S-PPJ-F Algorithm

The S-PPJ-B algorithm presented above exploits an upper bound on the number of unmatched objects between two

users in order to allow for early termination when comparing each pair of users. In the following, we present the S-PPJ-F algorithm that further increases efficiency by following a filter and refine strategy that concentrates the search on those pairs of users that are promising candidates, while pruning others that can not exceed the similarity threshold ϵ_u.

Algorithm 2: *S-PPJ-F* Algorithm

Input: D, U, ϵ_{doc}, ϵ_{loc}, ϵ_u
Output: Pairs of matched users R

1 $R \leftarrow \emptyset$
2 $G \leftarrow initialiseSTGridIndex(D, \epsilon_{loc})$
3 **foreach** $u \in U$ **do**
4 **foreach** $c \in C_u$ **do**
5 $T \leftarrow calculateTokens(u, c)$
6 **foreach** $c' \in G.getRelevantCells(c)$ **do**
7 **foreach** $t \in T$ **do**
8 **foreach** $u' \in G.getTokenUsers(c', t)$ **do**
9 $M^u_{u'}.add(c)$, $M^{u'}_{u'}.add(c')$
10 $G.addUser(u)$
11 **foreach** $u' \in M.keys()$ **do**
12 $m \leftarrow \sum_{c \in M^u_{u'}} |D^c_u| + \sum_{c' \in M^{u'}_{u'}} |D^{c'}_{u'}|$
13 $\bar{\sigma} \leftarrow \frac{m}{|D_u|+|D_{u'}|}$
14 **if** $\bar{\sigma} \geq \epsilon_u$ **then**
15 $\sigma \leftarrow PPJ\text{-}B(D_u, D_{u'}, G, \epsilon_{doc}, \epsilon_{loc}, \epsilon_u)$
16 **if** $\sigma \geq \epsilon_u$ **then**
17 $R.add(\langle u', u \rangle)$
18 **return** R

S-PPJ-F is outlined in Algorithm 2. It operates on top of a spatio-textual index structure that is constructed at runtime. In every iteration, the algorithm selects a new user u, searches for potential matches with the users that have been selected in previous steps, and updates the spatio-textual index with the objects in D_u.

The spatio-textual index is a dynamic grid enhanced with an inverted index for every cell. This list maintains for every token that appears in objects in a cell, the users that are associated with these objects. An example is depicted in Figure 3. The grid structure additionally maintains the objects associated with every user within a cell.

The search for matches follows the filter and refine principle. After a user u is selected, the algorithm traverses through every cell $c \in C_u$ which contains objects associated with u, and calculates the set of tokens T that appear in any one of these objects. This set is then utilized to identify candidate users in c and its surrounding cells (lines 6-9). Every user u' with objects that appear in one of these cells that at least one keyword from T is considered to be a candidate. $M^u_{u'}$ maintains cells that contain objects from u that potentially match (both spatially and textually) with objects from u'. Respectively, $M^{u'}_{u'}$ maintains the relevant for u'.

For every user u and candidate user u', the algorithm calculates an upper bound $\bar{\sigma}$ of their user similarity score (lines 12-13). This is performed by assuming that all of their objects which are contained in the same or adjacent cells match. Formally, $\bar{\sigma}$ is computed as follows:

$$\bar{\sigma} = \frac{\sum_{l \in M^u_{u'}} |D^l_u| + \sum_{l' \in M^{u'}_{u'}} |D^{l'}_{u'}|}{|D_u| + |D_{u'}|}.$$

If $\bar{\sigma} < \epsilon_u$, then this pair can be safely pruned. Otherwise, a refinement step follows, during which the PPJ-B algorithm is executed to identify whether the exact similarity score for the pair exceeds the user similarity threshold.

4.1.4 The S-PPJ-D Algorithm

In the following, we consider databases that are already partitioned by a data partitioning scheme. In particular, we consider data partitioning schemes induced by an R-tree structure combined with a textual index similar in fashion to the index outlined with respect to S-PPJ-F. The main difference is that instead of indexing grid cells, in this case, we index the leaf nodes of the R-tree.

S-PPJ-D implements a filter and refinement strategy similar to S-PPJ-F, based on a given data partitioning and a spatio-textual index I that is constructed at runtime. I maintains an entry for every leaf node l in the tree. This entry holds an inverted list that maps a token t U^l_t (i.e. users with objects in l that contain t). In addition, every leaf node l maintains a mapping between users and their objects within l, denoted by D^l_u. Finally, the intersections among the extended MBRs of the leaf nodes in the tree are precomputed by performing a spatial join using the process described in [8].

The filter step iterates over the leaf nodes L_u of a user u. For every leaf node l, it calculates the set of tokens T that appear in objects within l that are associated with u (i.e. D^l_u). These tokens are then used to probe the spatio-textual index and identify the candidate users that are associated with objects containing tokens from T. This is performed for each leaf node that intersects with the ϵ_{loc}-extended bounding box of l. To avoid duplicates, we only search for candidate users which are higher in the user ordering. M maintains for every candidate u' the leaf nodes of u' containing objects that can potentially match objects associated with user u $M^{u'}_{u'}$, as well as the leaf nodes of the relevant objects from u $M^u_{u'}$. S-PPJ-D calculates for every candidate u' a bound on the similarity score between u and u'. This is calculated by considering the extreme case in which all objects from $M^{u'}_{u'}$ and $M^u_{u'}$ match. The refinement step uses PPJ-D in order to calculate the exact similarity between candidate users.

Algorithm 3 outlines PPJ-D. PPJ-D leverages the spatio-textual index in combination with an appropriate leaf node traversal strategy in order to return the similarity score between two users. PPJ-D functions similar to PPJ-B for the context of a data-driven partitioning scheme. Given two users u_1 and u_2, two lists L_1 and L_2 are maintained for their leaf nodes ordered with respect to a predefined ordering (e.g. in ascending order of their ids). The algorithm proceeds iteratively, and selects the lowest (with respect to the ordering) unvisited leaf node l from L_1 and L_2.

Let user u be the user from which the element was selected, and u' the other user. The index is used to identify every leaf node l' that is spatially relevant to l, and contains objects from u'. Spatially relevant leaf nodes are nodes with intersecting ϵ_{loc}-extended MBRs. For every l' we execute PPJoin to identify the exact similarity between the objects D^l_u and $D^{l'}_{u'}$. This is performed by focusing only on objects that belong within the intersection A of the ϵ_{loc}-extended MBRs of l and l' (lines 11-12, 18-19). This optimisation is based on the observation that objects which are not contained in A do not satisfy the spatial threshold ϵ_{loc}.

PPJ-D follows a similar pruning strategy with PPJ-B. The

Algorithm 3: PPJ-D Algorithm

Input: D_{u_1}, D_{u_2}, I, ϵ_{doc}, ϵ_{loc}, ϵ_u
Output: Similarity score for users u_1, u_2

1 $\beta \leftarrow (1-\epsilon_u) \cdot (|D_{u_1}| + |D_{u_2}|)$
2 $J \leftarrow \emptyset$ // joined objects
3 $L_1 \leftarrow I.getLeafs(u_1)$ // sorted
4 $L_2 \leftarrow I.getLeafs(u_2)$
5 $i_1 \leftarrow 0$, $i_2 \leftarrow 0$
6 $t \leftarrow 0$
7 **while** $i_i < |L_1|$ ***or*** $i_2 < |L_2|$ **do**
8 **if** $L_1[i_i] \leq L_2[i_2]$ **then**
9 **foreach** $l_2 \in I.getRelevantLeafs(l_1)$ **do**
10 **if** $l_2 \geq l_1$ ***and*** $l_2 \in L_2$ **then**
11 $A \leftarrow I.extend(l_1, \epsilon_{loc}) \cap I.extend(l_2, \epsilon_{loc})$
12 $PPJoin(D^{l_1}_{u_1} \cap A, D^{l_2}_{u_2} \cap A, J)$
13 $t \leftarrow t + |D^{l_1}_{u_1}|$
14 **else if** $L_{u_2}[i_2] \leq L_1[i_1]$ **then**
15 $l_2 \leftarrow L_2[i]$
16 **foreach** $l_1 \in I.getRelevantLeafs(l_2)$ **do**
17 **if** $l_1 > l_2$ ***and*** $l_1 \in L_1$ **then**
18 $A \leftarrow I.extend(l_1, \epsilon_{loc}) \cap I.extend(l_2, \epsilon_{loc})$
19 $PPJoin(D^{l_1}_{u_1} \cap A, D^{l_2}_{u_2} \cap A, J)$
20 $t \leftarrow t + |D^{l_2}_{u_2}|$
21 **if** $t - |J| > \beta$ **then**
22 **return** 0
23 **if** $L_1[i_i] <= L_2[i_2]$ **then** $i_1 \leftarrow i_1 + 1$
24 **if** $L_2[i_2] <= L_1[i_1]$ **then** $i_2 \leftarrow i_2 + 1$
25 $\sigma \leftarrow |J|/(|D_{u_1}| + |D_{u_2}|)$
26 **if** $\sigma \geq \epsilon_u$ **then return** σ
27 **else return** 0

objects of every leaf node for user u are evaluated against every potential candidate from $D_{u'}$ that falls within a leaf node that is higher in the given ordering. Therefore, after an iteration that visits an object, candidate matches from leaf nodes, both higher and lower in the ordering, are considered. This observation is the basis of a pruning step (lines 21-22) that calculates the number of objects $t-|J|$ that are already found to fail to satisfy the thresholds. If this number is lower that a computed bound, the search is pruned since the users fail to satisfy the user similarity threshold.

4.2 Algorithms for top-k STPSJoin

Next, we extend our methods to support the top-k `STPSJoin` query. The main intuition behind our approach is that the algorithm must keep track of the top-k pairs identified thus far, and utilise the exact user similarity score of the kth best pair to update the user similarity threshold.

4.2.1 TOPK-S-PPJ-F

Algorithm 4 outlines `TOPK-S-PPJ-F`, which modifies `S-PPJ-F` for the purposes of the top-k `STPSJoin` query. The main modifications with respect to `S-PPJ-F` relate to the maintenance of intermediate results and the update of the user similarity threshold. Results are stored in a fixed capacity priority queue of size k, which is updated whenever a pair that is better than the kth pair in the queue is identified. The user similarity threshold ϵ_u is set as the similarity score of the kth best pair in the queue. Accordingly, the

Algorithm 4: *TOPK-S-PPJ-F* Algorithm

Input: D, U, ϵ_{doc}, ϵ_{loc}, k
Output: Top-k Pairs of matched users R

1 $R \leftarrow \emptyset$
2 $\epsilon_u \leftarrow -1$
3 $G \leftarrow initialiseSTGridIndex(D, \epsilon_{loc})$
4 **foreach** $u \in sorted(U)$ **do**
5 **foreach** $c \in C_u$ **do**
6 $T \leftarrow calculateTokens(u, c)$
7 **foreach** $c' \in G.getRelevantCells(c)$ **do**
8 **foreach** $t \in T$ **do**
9 **foreach** $u' \in G.getTokenUsers(c', t)$ **do**
10 $M^u_{u'}.add(c)$, $M^{u'}_{u'}.add(c')$
11 $G.addUser(u)$
12 **foreach** $u' \in M.keys()$ **do**
13 $m \leftarrow \sum_{c \in M^u_{u'}} |D^c_u| + \sum_{c' \in M^{u'}_{u'}} |D^{c'}_{u'}|$
14 $\bar{\sigma} \leftarrow \frac{m}{|D_u|+|D_{u'}|}$
15 **if** $\bar{\sigma} > \epsilon_u$ **then**
16 $\sigma \leftarrow PPJ\text{-}B(D_u, D_{u'}, G, \epsilon_{doc}, \epsilon_{loc}, \epsilon_u)$
17 **if** $\sigma > \epsilon_u$ **then**
18 $R.update(\langle u', u \rangle)$
19 **if** $|R| = k$ **then**
20 $\epsilon_u \leftarrow R.getTail()$
21 **return** R

threshold ϵ_u is updated whenever a new pair is introduced in the results queue (lines 18-20). The user similarity threshold is used in the filtering phase of the algorithm in a similar manner with `S-PPJ-F`. The same principle can be straightforwardly applied to `S-PPJ-D`. Pseudocode for the resulting algorithm is omitted due to lack of space.

`TOPK-S-PPJ-F` orders users in an ascending order of the size of their object-sets. This strategy is based on the observation that the treatment of users with larger object-sets requires more computations than the evaluation of users with fewer objects. By the time the algorithm reaches the most computationally demanding users, the user similarity threshold has been updated to reflect the best pairs identified so far, increasing the possibility that pairs that do not belong to the top-k result set are filtered out.

4.2.2 TOPK-S-PPJ-S

`TOPK-S-PPJ-S` operates similarly with `TOPK-S-PPJ-F`. However, it uses a heuristic strategy in order to decide the order by which users are evaluated. User objects are placed in a spatial grid and each cell in the grid c is given a score by counting the amount of users whose object-sets belong to c or its adjacent cells. Users are then assigned a score by summing, for every object o associated with them, the score of the cell that o is contained in. Formally, cell scores s_c are calculated as follows:

$$s_c = |\cup_{c' \in G.getRelevantCells(c)} G.getUsers(c')|$$

where G is the spatial grid, c is a cell in the grid, $G.getUsers$ returns the users with objects in c and $G.getRelevantCells(c)$ returns the cells that are adjacent to c (including c).

Accordingly, users are assigned scores s_u according to the

following formula:

$$s_u = \sum_{o \in D_u} s_{o_c}$$

D_u denotes the object-set for user u and o_c describes the cell that object o is located in.

Therefore the rationale behind `TOPK-S-PPJ-S` is to start the search with users whose objects are placed in popular areas. This strategy aims at quickly identifying high scoring pairs, in order to increase the user similarity threshold quickly, and improve the efficiency of the filtering step.

4.2.3 TOPK-S-PPJ-P

The `TOPK-S-PPJ-P` algorithm introduces an additional filtering step. Users are selected in ascending order of the size of their object-sets. For every user u, we calculate an upper bound on the similarity score between u and any user u' that was selected in a previous iteration. To do so, we identify the objects from D_u that match with any object from $D_{U'}$, i.e. the union of the objects of every user that was selected in previous iterations. This is denoted as $M(D_u, D_{U'})$. This allows the calculation of an upper bound on $\sigma(u, u')$ for every u' that was selected prior to u. Formally, this bound is calculated as follows:

$$\bar{\sigma}_u = \frac{|\bigcup_{u' \in U'} M(D_u, D_{u'})| + \max_{u' \in U'} |D_{u'}|}{|D_u| + \max_{u' \in U'} |D_{u'}|}$$

If the users are selected in an ascending order of the size of their object-sets, we show that $\bar{\sigma}_u$ is an upper bound on the similarity score between u and a user u' with fewer or equal objects.

LEMMA 2. *Let a user u and a set of users U'. If for every user $u' \in U'$ $|D_{u'}| \leq |D_u|$, then it holds that $\sigma(u, u') \leq \bar{\sigma}_u$.*

PROOF. Let for any user $u' \in U$, $m_u = |\bigcup_{u'' \in U'} M(D_u, D_{u''})|$, $m_u^{u'} = |M(D_u, D_{u'})|$, $m_{u'}^{u} = |M(D_{u'}, D_u)|$, $d_u = |D_u|$ and $d_{U'} = \max_{u'' \in U'} |D_{u'}|$. Then, since $m_u \geq m_u^{u'}$ and $d_{u'} \geq m_{u'}$ it holds that

$$\frac{m_u + d_{u'}}{d_u + d_{u'}} \geq \sigma(u', u)\,.$$

We show that:

$$\bar{\sigma}_u = \frac{m_u + d_{u''}}{d_u + d_{u''}} \geq \frac{m_u + d_{u'}}{d_u + d_{u'}}\,.$$

$$\frac{m_u + d_{u''}}{d_u + d_{u''}} \geq \frac{m_u + d_{u'}}{d_u + d_{u'}} \Rightarrow$$
$$m_u \cdot d_u + m_u \cdot d_{u'} + d_{u''} \cdot d_u + d_{u''} \cdot d_{u'} \geq$$
$$m_u \cdot d_u + m_u \cdot d_{u''} + d_{u'} \cdot d_u + d_{u'} \cdot d_{u''} \Rightarrow$$
$$(d_{u''} - d_{u'}) \cdot d_u \geq (d_{u''} - d_{u'}) \cdot m_u, .$$

This holds since $d_{u''} \geq d_{u'}$ and $d_u \geq m_u$. □

In order to avoid the computation of exact similarity scores among user objects, and speed up the bound calculation process, we follow the same principle with the filtering step from the `S-PPJ-F` algorithm. We utilise the spatio-textual index described in Figure 3 and place in $M(D_u, D_{U'})$ every object with a token that appears (due to a previously selected user) in the same or adjacent cell. This process provides a fast estimation of the $\bar{\sigma}_u$ bound. Since this process overestimates, the resulting score is still an upper bound on the actual user similarity score and can be used to prune the search space. This process yields relaxed bounds, which are irrelevant for a fixed user similarity threshold (as in the case of `STPSJoin`). However, it is useful with respect to the top-k algorithms that quickly increase the user similarity threshold.

5. EXPERIMENTAL EVALUATION

Next, we present our experimental evaluation of the proposed algorithms. We first describe the datasets used and the parameters involved, and then we present the results.

5.1 Experimental Setup

Datasets. We have used three real-world datasets of spatio-textual web objecs for our experiments. The *Flickr* dataset is derived from the Flickr Creative Commons dataset provided by Yahoo [35]. The whole dataset contains about 99.3 million images, about 49 million of which are geotagged. For our experiments, we concentrate on objects from the geographical boundaries of London, UK and we filtered out images that do not contain coordinates or tags as well as those that are created by stationary users. The resulting dataset contains 11,306 users and 1,116,348 objects. The *GeoText* dataset [18] is a geotagged microblog corpus available online.[1] It comprises 377,616 posts by 9,475 different users within the US. Finally, the *Twitter* dataset is a collection of geotagged tweets from the geographical area of London, UK, that we have collected and is part of the dataset used in [17]. It contains 9,724,579 tweets generated by 40,000 different users in 2014.

The NLTK toolkit[2] was employed to identify named entities from the text associated with the objects. The extracted named entities were used in combination with other related information, such as tokens, hashtags and mentions, as keywords associated with the respective objects. The characteristics of the three datasets are summarized in Table 1.

Evaluation measures and parameters The purpose of the experimental evaluation is to compare the performance of the proposed algorithms in terms of the execution time in different settings. For the case of the `STPSJoin` query, we investigate the effect of the following parameters: (a) the dataset size N in terms of number of users, (b) the query thresholds for spatial distance (ϵ_{loc}), textual similarity (ϵ_{doc}) and user similarity (ϵ_u) and (c) the *fanout* parameter of the R-tree structure. The effect of these parameters on the results of the `STPSJoin` query in the experimental datasets are described in Table 2. The largest deviation is observed on the Flickr dataset. This is consistent with the nature of this dataset, since popular POIs are often described using similar textual descriptions as well as photographs depicting these POIs are usually captured in nearby locations. On the contrary, the other datasets contain tweets, which are significantly more diverse both with respect to spatial locations and textual descriptions. For the top-k `STPSJoin` query, we investigate the effect of the parameter k on execution time.

All algorithms were implemented in Java, and the experiments were executed on a machine with an Intel Core i5 2400 CPU and 16GB RAM, running on Ubuntu Linux. During the experiments, 15GB of memory were allocated to the JVM. All plots report running time in a logarithmic scale.

[1] `http://www.ark.cs.cmu.edu/GeoText/`
[2] `http://www.nltk.org/`

Dataset	Objects	Users	Tokens per Object	Objects per Token	Objects per User
Twitter	9,724,579	40,000	2.08 (1.43)	6.25 (141.80)	243.11 (344.86)
Flickr	1,116,348	11,306	8.04 (8.15)	26.41 (1,191.09)	98.73 (419.92)
GeoText	165,733	9,461	1.64 (1.01)	3.53 (39.36)	17.52 (12.99)

Table 1: Experimentation datasets, number of objects and users, and mean (standard deviation) for descriptive metrics.

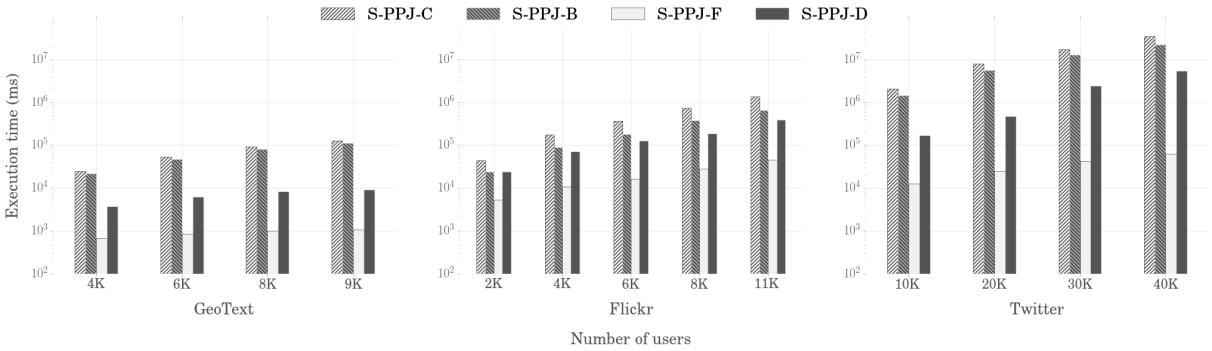


Figure 4: Scalability results for the GeoText, Flickr and Twitter datasets (parameter defaults: GeoText: $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.3$, $\epsilon_u = 0.3$; Flickr $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.6$, $\epsilon_u = 0.6$; Twitter: $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.4$, $\epsilon_u = 0.4$).

	GeoText	Flickr	Twitter
Scalability	27.00 (8.51)	54.20 (46.22)	13.50 (6.54)
Tuning	18.00 (36.90)	326.00 (633.89)	14.14 (9.98)

Table 2: Mean (std-dev) of result-set sizes

5.2 Scalability

The scalability experiments evaluate the performance of our methods in datasets of different sizes. We divided the Twitter, Flickr and GeoText datasets for variable numbers of users. The resulting datasets range from 4,000 users with 72,094 objects to 40,000 users with 9,724,579 objects. Different parameter values are used for different datasets, in order to account for different sizes and token selectivity across datasets. Lower thresholds are set for the GeoText dataset in order to avoid empty result sets, whereas higher thresholds are set for the Flickr dataset to account for the higher similarity between user objects. This is due to the fact that a large amount of Flickr photos represent popular POIs that are described by similar textual content and are geo-located close to the location of the corresponding POIs.

Figure 4 shows the scalability evaluation results. The results clearly show that `S-PPJ-F` outperforms the other methods by several orders of magnitude. This is consistent for all datasets, irrespective of size. The efficiency of `S-PPJ-F` compared to the other approaches is attributed to the effect of the filter and refinement scheme, in combination with the suitability of the dynamic grid partitioning over the objects. The grid partitioning is tailor made to the spatial threshold parameter ϵ_{loc}, which allows the search for matching objects to be limited exclusively in adjacent cells. Additionally, the inverted lists maintained within each cell of the grid, allow the effective filtering of candidate user pairs associated with spatially similar, but textually diverse, objects.

The performance of `S-PPJ-B` does not compare favourably against `S-PPJ-F`. This result is expected since `S-PPJ-F` builds on `S-PPJ-B` by leveraging the filter and refinement scheme. Nevertheless, the comparison between `S-PPJ-B` and `S-PPJ-C` allows the evaluation of the early termination strategy, as well as the traversal mechanism, differentiating `S-PPJ-B` from `S-PPJ-C`. The results indicate that `S-PPJ-B` offers a consistent improvement in execution time compared to `S-PPJ-C`, confirming that the proposed techniques manage to prune the search space for similarity search among two point sets.

Finally, the results show that `S-PPJ-D` outperforms the baseline methods, but it is not comparable to the grid-based `S-PPJ-F`, which follows the same principles. The discrepancy in execution time can be attributed to the use of different spatial indexes. The data driven-partitioning imposed by the R-tree is independent of the spatial threshold given as a parameter to the `STPSJoin` query. As a result, the imposed partitioning leads to an ineffective division of the database. Inspection of the performance of `S-PPJ-D` shows that both partition size and overlap may lead to subpar performance, since objects within large partitions tend to be spatially irrelevant, and overlaps require the evaluation of multiple join operations. We revisit this issue in Section 5.4.

5.3 Effect of similarity thresholds

In the following experiments, we vary the parameters and evaluate the proposed algorithms for different combinations of textual, spatial and user similarity thresholds. Similar to the scalability experiments, different ranges in threshold values are used across datasets.

Figure 5 presents the results. We observe that the dominant parameter is the spatial threshold ϵ_{loc}. High values on ϵ_{loc} result in significantly higher execution times. This is particularly obvious for the Flickr and Twitter datasets, which contain significantly larger amounts of objects. When the spatial distance threshold reaches metropolitan level distances, the majority of the objects fall into adjacent partitions. As a result, the filtering step of `S-PPJ-F` and `S-PPJ-D` returns a high number of candidates. In these cases, the overhead imposed by the additional indexing maintained by `S-PPJ-F` and `S-PPJ-D` is apparent. We observe a peak in the case of `S-PPJ-D`, especially with respect to the Flickr

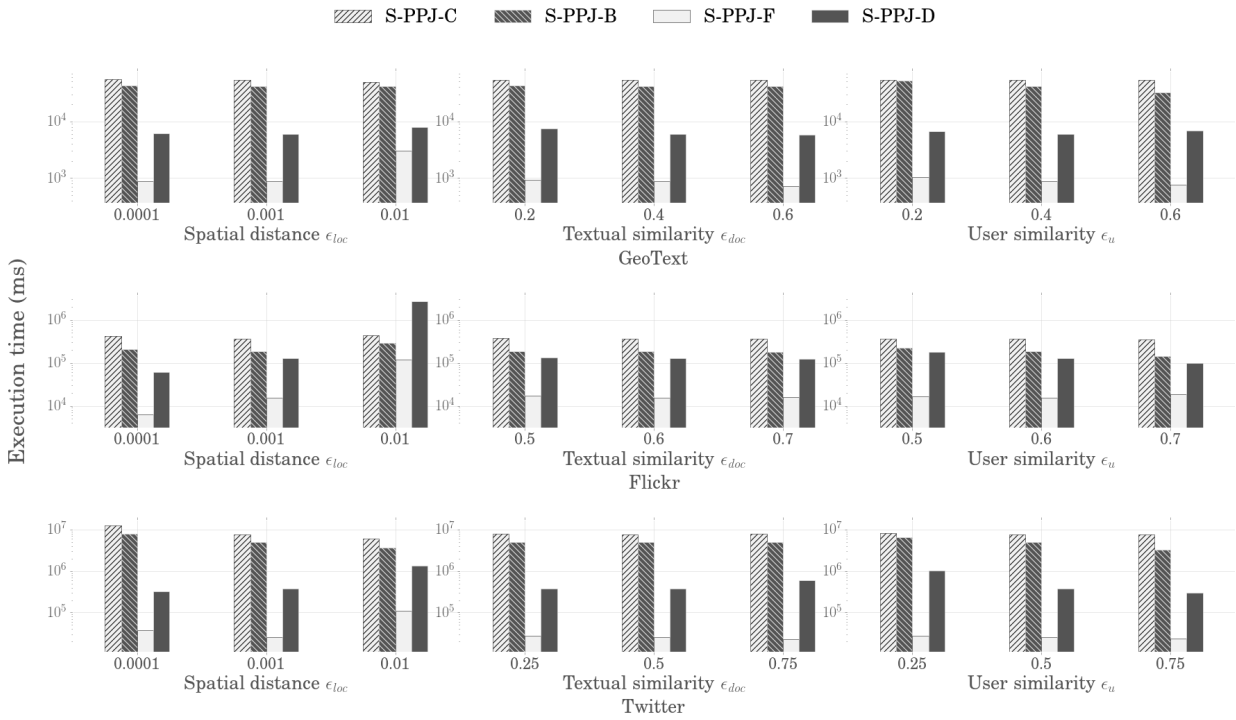


Figure 5: Results for varying similarity thresholds (GeoText: 6,000 users, 107,941 objects; Flickr: 6,000 users, 597,008 objects; Twitter: 20,000 users, 4,988,090 objects).

dataset. In this case, inspection shows that the R-tree partitioning does not manage to result in an efficient partition of the object database.

This does not apply for GeoText, mainly due to the fact that the objects in GeoText are scattered in the significantly larger area of the whole of USA. The results show that the proposed pruning strategies are highly functional in combination with a grid-based partitioning scheme. `S-PPJ-F` outperforms the other methods in every scenario, and apart from the case of the Flickr dataset with $\epsilon_{loc} = 0.01$, its performance is independent of the parameter values.

5.4 Effect of Fanout on S-PPJ-D

An important parameter for data partitioning schemes based on R-trees is the *fanout* parameter. This parameter is associated with the number of objects that reside in a node of the R-tree. The effect of the fanout parameter on the performance of `S-PPJ-D` is twofold. First of all, `S-PPJ-D` executes a spatial distance join in order to identify spatial relations among the leaf nodes of the tree, which are treated by the algorithm as spatial data partitions. Since the fanout parameter affects both the depth and the breadth of the R-tree, it also affects the performance of the spatial join. Second, `S-PPJ-D` is built on top of the partitioning imposed by the leaf nodes. Therefore, the fanout affects both the number and the size of leaf nodes, which are relevant to `S-PPJ-D`.

In order to experimentally evaluate the effects of the fanout parameter, experiments with values ranging from 50 to 250 were conducted. The results are shown in Figure 6. The results verify that `S-PPJ-D` is sensitive to the fanout value. Even though no single fanout value achieves the best results in all datasets, we observe that an appropriate fanout value for `STPSJoin` queries falls within the range of 100 to 200.

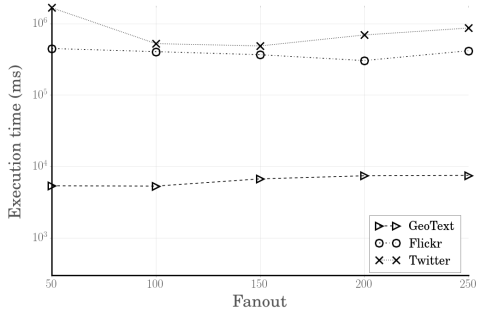


Figure 6: Tuning the R-Tree fanout parameter.

5.5 Evaluation of top-k STPSJoin

In the following, we evaluate the proposed algorithms for the top-k `STPSJoin` query. We vary the result size k in order to study the behaviour of the algorithms. Figure 7 shows the results of the experiments. The baseline `TOPK-S-PPJ-F` is competitive and is the better performing algorithm in the Flickr dataset. The poor performance of `TOPK-S-PPJ-S` shows that the simple ordering of the users based exclusively on the size of their object-sets is more efficient than the statistical approach that it follows, compared to the overhead that the additional computation imposes. `TOPK-S-PPJ-P` exploits an additional pruning step and offers a better performance in the cases of GeoText and Twitter. In the case of Flickr, while it is outperformed by `TOPK-S-PPJ-F`, it remains competitive. This is due to the fact the the Flickr dataset contains objects with very high similarity, mainly because of the nature of the Flickr service (ie. people describe popular places with nearly the same keywords). The

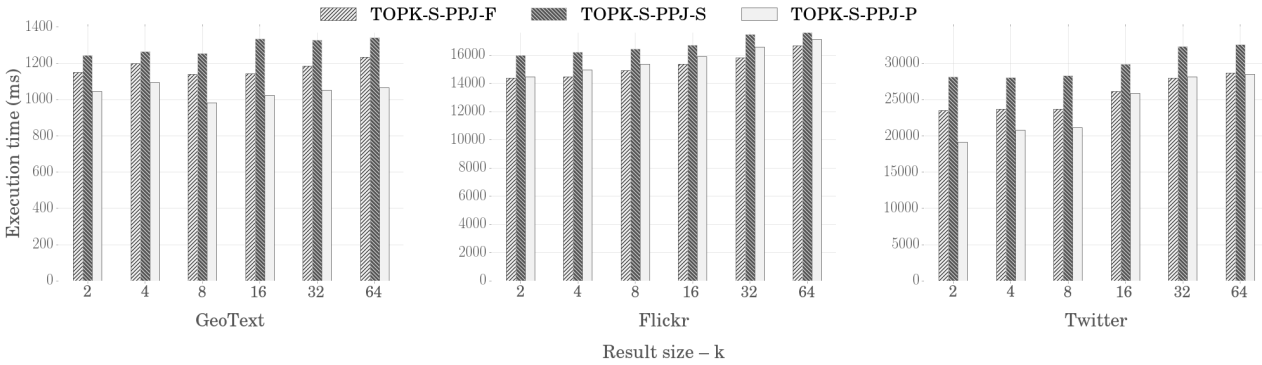


Figure 7: Results for the top-k STPSJoin algorithms on GeoText, Flickr and Twitter datasets with varying k (GeoText: $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.3$, $\epsilon_u = 0.3$; Flickr $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.6$, $\epsilon_u = 0.6$; Twitter: $\epsilon_{loc} = 0.001$, $\epsilon_{doc} = 0.4$, $\epsilon_u = 0.4$).

very high textual similarity between objects leads to high user similarities. Therefore, the additional filtering step of TOPK-S-PPJ-P cannot disqualify large numbers of user pairs.

5.6 Parameter Tuning

The STPSJoin query requires ϵ_{loc}, ϵ_{doc}, and ϵ_u to be provided as input. The values of these thresholds define what is "near" in terms of spatial, textual and user similarity, and are determined by the nature of the data and task in hand. In order to tackle situations in which there is no prior knowledge that can be used to determine the values of these thresholds, we present an automated process in order to discover sensible thresholds. In this case, the necessary input is an acceptable result set size.

The tuning algorithm is initialised with relaxed initial thresholds. Our experiments show that these can be predefined values regardless of the dataset. The only requirement is that they are relaxed enough to guarantee a result-set larger than the input value. Threshold steps are calculated as fractions of the initial values.

The algorithm follows a greedy strategy. Initially, it executes S-PPJ-F using the starting thresholds and populates a result-set. Then, the process traverses the parameter combination space in a depth-first manner. At any given step the algorithm selects probabilistically which parameter to tighten (an alternative strategy is to modify the least modified threshold). Tightening the parameters monotonically decreases the results-set that was the outcome of the previous step. As a result, S-PPJ-F is not executed again for the different parameters. Instead, PPJ-C is used to identify which pairs from the previous step adhere to the new thresholds. If the result set size reaches the desired value, the algorithm stops and the current threshold values are returned. If a step brings about thresholds that yield no results, the process backtracks to the previous step, and an alternative threshold is tightened.

	S-PPJ-F	Tuning		
Result size		5	25	50
GeoText	2,229	145 (8)	124 (4)	110 (3)
Flickr	24,363	738 (23)	693 (17)	1,066 (10)
Twitter	82,412	1,278 (10)	3,085 (7)	1,439 (2)

Table 3: Parameter tuning including S-PPJ-F time and tuning time in ms (number of iterations) for varying result-sets.

Table 3 shows the time required for parameter tuning after the initial execution of S-PPJ-F. Initial thresholds were the minimum thresholds used in Section 5.3, and the datasets were those with the minimum number of users used in Section 5.2. It is worth noting that the initial running time of S-PPJ-F consumes a significant amount of the overall time.

5.7 Summary

Our experimentation verifies the superiority of the proposed algorithms for the treatment of the STPSJoin query, in terms of execution time for all datasets used. The pruning strategy employed by S-PPJ-F manages to significantly boost the algorithm's performance. Furthermore, the S-PPJ-D algorithm which induces data partitioning is efficient enough to be considered as a viable choice in cases when the data are already partitioned with an R-tree (or any other data partitioning method). Our experimentation shows that S-PPJ-F can be directly modified to efficiently handle the top-k STPSJoin query variant. Nevertheless, we propose an additional pruning strategy in top-k STPSJoin that performs even better with datasets of lower degrees of similarity. Even in the case of the Flickr dataset, which does not fall into this category, TOPK-S-PPJ-P achieves competitive results. The experimental analysis is conducted on three real datasets of varied size (the two of them are publicly available) and different parameter settings have been examined in order to reach to the optimum configurations. The results show that the algorithms scale well in very large databases and can therefore be used effectively in real-world scenarios.

6. CONCLUSIONS

This paper studies the problem of similarity search on spatio-textual point sets. We formally define this problem as STPSJoin and present its top-k variant. STPSJoin queries identify pairs of similar users, with respect to web documents such as tweets and photographs associated with these users.

In order to efficiently process (top-k) STPSJoin queries, we propose algorithms that leverage different spatio-textual indexes, and integrate early termination pruning mechanisms with filter and refinement approaches. We conducted large-scale experiments on real-world datasets for multiple values on the problem parameters. The better performing algorithm S-PPJ-F is orders of magnitude more efficient in terms of execution time than the baseline methods. Finally, S-PPJ-D shows improvement over the baseline methods for

the case of data-driven partitioned databases, even though it is significantly outperformed by `S-PPJ-F`. For the case of the top-k `STPSJoin` query, the `TOPK-S-PPJ-P` algorithm offers the best results in the majority of the datasets, but also remains competitive in the case of the Flickr dataset, which contains significantly larger amounts of similar spatio-textual objects.

In the future, we plan to focus on distributed architectures in order to further enhance the efficiency of our methods. Furthermore, we intend to integrate additional characteristics in `STPSJoin` queries, which are often associated with web objects, such as temporal information.

Acknowledgements

This work was partially supported by the EU Project City.Risks (H2020-FCT-2014-653747).

7. REFERENCES

[1] M. D. Adelfio, S. Nutanong, and H. Samet. Searching web documents as location sets. In *ACM SIGSPATIAL*, 2011.
[2] M. D. Adelfio, S. Nutanong, and H. Samet. Similarity search on a large collection of point sets. In *ACM SIGSPATIAL*, 2011.
[3] J. Ballesteros, A. Cary, and N. Rishe. Spsjoin: Parallel spatial similarity joins. In *ACM SIGSPATIAL*, 2011.
[4] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel. Recommendations in location-based social networks: a survey. *GeoInformatica*, 19(3):525–565, 2015.
[5] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.
[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
[7] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, pages 1–12, 2012.
[8] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, 1993.
[9] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, 2011.
[10] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.
[11] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.
[12] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, 2006.
[13] 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.
[14] J. Cranshaw, E. Toch, J. I. Hong, A. Kittur, and N. M. Sadeh. Bridging the gap between physical location and online social networks. In *UbiComp*, 2010.
[15] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, 2008.
[16] P. DeScioli, R. Kurzban, E. N. Koch, and D. Liben-Nowell. Best friends alliances, friend ranking, and the myspace social network. *Perspectives on Psychological Science*, 6(1):6–8, 2011.
[17] H. Efstathiades, D. Antoniades, G. Pallis, and M. D. Dikaiakos. Identification of key locations based on online social network activity. In *ASONAM*, 2015.
[18] J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing. A latent variable model for geographic lexical variation. In *EMNLP*, 2010.
[19] T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.
[20] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. Seal: Spatio-textual similarity search. *PVLDB*, 5(9):824–835, May 2012.
[21] A. Goel, A. Sharma, D. Wang, and Z. Yin. Discovering similar users on twitter. In *MLG*, 2013.
[22] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
[23] E. H. Jacox and H. Samet. Spatial join techniques. *TODS*, 32(1):7, 2007.
[24] Y. Jiang, G. Li, J. Feng, and W.-S. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.
[25] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W. Ma. Mining user similarity based on location history. In *ACM SIGSPATIAL*, 2008.
[26] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.
[27] S. Liu, G. Li, and J. Feng. Star-join: Spatio-textual similarity join. In *CIKM*, 2012.
[28] S. Liu, G. Li, and J. Feng. A prefix-filter based method for spatio-textual similarity join. *TKDE*, 26(10):2354–2367, 2014.
[29] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, 2013.
[30] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *SSTD*. Springer-Verlag, 2001.
[31] J. Ramon and M. Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.
[32] J. Rao, J. Lin, and H. Samet. Partitioning strategies for spatio-textual similarity join. In *BigSpatial*, 2014.
[33] J. a. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, 2011.
[34] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD*, 2004.
[35] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.
[36] S. Vaid, C. B. Jones, H. Joho, and M. S. Spatio-textual indexing for geographical search on the web. In *SSTD*, 2005.
[37] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*, pages 85–96, 2012.
[38] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *TODS*, 36(3):15:1–15:41, Aug. 2011.
[39] X. Xiao, Y. Zheng, Q. Luo, and X. Xie. Finding similar users using category-based location history. In *ACM SIGSPATIAL*, 2010.
[40] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, 2009.
[41] D. Zhang, B. C. Ooi, and A. K. Tung. Locating mapped resources in Web 2.0. In *ICDE*, 2010.
[42] Y. Zheng, L. Zhang, Z. Ma, X. Xie, and W. Ma. Recommending friends and locations based on individual location history. *TWEB*, 5(1):5, 2011.
[43] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, 2005.