The VLDB Journal (2018) 27:297–320
https://doi.org/10.1007/s00778-018-0498-5

REGULAR PAPER

Spatio-textual user matching and clustering based on set similarity joins

Alexandros Belesiotis[1] · Dimitrios Skoutas[1] · Christodoulos Efstathiades[2] · Vassilis Kaffes[1] · Dieter Pfoser[3]

Received: 29 June 2017 / Revised: 19 December 2017 / Accepted: 23 February 2018 / Published online: 6 March 2018

Abstract
This paper addresses the problem of matching and clustering users based on their geolocated posts. Individual posts are matched according to spatial distance and textual similarity thresholds. Then, user similarity is defined as the ratio of their posts that match each other. Based on these criteria, we introduce efficient algorithms for identifying pairs of matching users in a large dataset, as well as for computing the top-k matching pairs. We then proceed to identify spatio-textual user clusters. For this purpose, we use the Louvain method for community detection. Our algorithms operate on a user graph where edge weights represent spatio-textual user similarities. Since the exact user similarity graph can be prohibitively expensive to compute, we exploit our previous algorithms to derive efficient methods that reduce execution time both by avoiding to compute exact similarity scores and by reducing the number of similarity calculations performed. The presented solution allows a trade-off between computation time and quality of detected clusters. The proposed algorithms are evaluated using three real-world datasets.

Keywords Spatio-textual join · Set similarity join · Spatio-textual clustering

1 Introduction

Social networking sites, such as Twitter, Flickr, Facebook, or Foursquare, attract millions of users generating a multitude of content daily. According to various online statistics (August 2016), Twitter has more than 300 million monthly active users, generating more than 500 million tweets per day. Moreover, more than 80% of these users access Twitter via a mobile device. This user-generated content comprises primarily textual information (e.g., status updates, short messages, tags). In addition, an increasingly large portion of it is geotagged, being produced via GPS-enabled devices, or can be geotagged via available geocoding tools and services. Combining this textual and spatial information can reveal valuable insights about user preferences, interests, behaviors, and habits.

Finding users with similar behavior is fundamental for various types of analyses. In this work, we focus on identifying similar users based on their spatio-textual "footprint", i.e., the location and content of their geotagged posts. Some users may often visit the same or nearby places but may be interested in different things; vice versa, other users may share common topics and interests but may be situated far apart. Hence, by considering their combined spatio-textual similarity, we aim at discovering users that match in both criteria, i.e., users that tend to visit the same or nearby locations and generate similar posts.

An example is presented in Fig. 1. This depicts a set of 8 spatio-textual objects (e.g., geolocated tweets), denoted as o_1 to o_8, belonging to 3 different users, u_1, u_2 and u_3. Each object is associated with a label indicating the user it belongs to and the keywords it contains. Assume that two posts match if they contain at least one common keyword and if, additionally, the spatial distance of their locations is below a given

✉ Dimitrios Skoutas
dskoutas@imis.athena-innovation.gr

Alexandros Belesiotis
abelesiotis@imis.athena-innovation.gr

Christodoulos Efstathiades
C.Efstathiades@euc.ac.cy

Vassilis Kaffes
vkaffes@imis.athena-innovation.gr

Dieter Pfoser
dpfoser@gmu.edu

[1] IMIS, R.C. Athena, Athens, Greece

[2] European University Cyprus, Nicosia, Cyprus

[3] George Mason University, Fairfax, USA

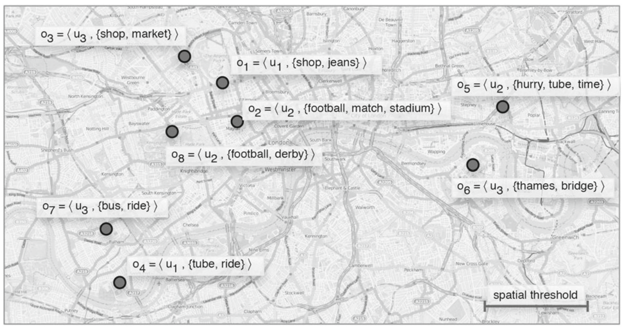

$o_3 = \langle u_3$, {shop, market} ⟩
$o_1 = \langle u_1$, {shop, jeans} ⟩
$o_5 = \langle u_2$, {hurry, tube, time} ⟩
$o_2 = \langle u_2$, {football, match, stadium} ⟩
$o_8 = \langle u_2$, {football, derby} ⟩
$o_6 = \langle u_3$, {thames, bridge} ⟩
$o_7 = \langle u_3$, {bus, ride} ⟩
$o_4 = \langle u_1$, {tube, ride} ⟩
spatial threshold

Fig. 1 Example of geolocated posts by three users

threshold, as illustrated on the map. Users u_1 and u_2 have posts in nearby locations (o_1, o_2), but refer to different topics (do not share any common keywords). In contrast, users u_1 and u_3 are more similar spatio-textually, as they have two pairs of posts, (o_1, o_3) and (o_4, o_7), that are both spatially close and contain a common keyword.

In this work, we do not rely on social links between users (e.g., friends or followers). This differentiates our problem setting from works that aim at discovering communities on the users' social graph (e.g., [14,21,33]). Instead, the user graph in our case is based on spatio-textual user similarities rather than social connections. We consider social relationships among users as an orthogonal criterion. In several applications, such as in geomarketing, mobile advertisement, or urban planning, it is important to identify groups of users with similar spatio-textual characteristics, regardless of the existence of social relations among them.

We model the users' posts as spatio-textual objects, each comprising a location and a set of keywords (e.g., tags attached to a photograph, hashtags contained in a tweet, or named entities extracted from a text message). Accordingly, we represent each user by the set of her geotagged posts. These may span the whole lifetime of the user or could be restricted to a specific time period. Based on these, we define the spatio-textual similarity of two users. Then, we address two problems: (a) how to efficiently compute pairs of matching users and (b) how to efficiently partition users into "communities" according to their spatio-textual similarities.

Queries on spatio-textual data have been extensively studied, considering several variants, including Boolean range queries, top-k queries, and k-nearest neighbor queries [12]. A typical application of such queries is location-based search, where the user provides her location and a set of keywords, and the query returns a set of objects that contain one or more of these keywords and are filtered or ranked by proximity to the user's location. The proposed solutions rely on hybrid spatio-textual indices. Efficient algorithms for spatio-textual similarity joins have also been proposed [8]. In this case, given a dataset of spatio-textual objects, and two thresholds δ and τ, the join returns those pairs of objects having spatial distance below δ and textual similarity above τ. Typical applications include, for example, matching Points of Interest from different sources or matching geotagged photographs from different users. However, all above works match pairs of spatio-textual objects. Instead, in our setting, each user is represented by a *set* of such spatio-textual objects. This makes the problem more complex and raises the need for new algorithms to address it efficiently.

We model the task of finding similar users based on their geotagged posts as a Spatio-Textual Point-Set Similarity Join (`STPSJoin`) query. First, we present a baseline algorithm, `S-PPJ-C`, which extends the `PPJ-C` algorithm [8], designed for individual spatio-textual objects, to operate on sets of such objects. Then, we propose two optimized algorithms, `S-PPJ-B` and `S-PPJ-F`. The former applies an early termination condition, while the latter uses a filter-and-refinement strategy to drastically prune the search space. These algorithms significantly reduce the number of comparisons required, both in terms of pairs of users and in terms of individual spatio-textual objects for each candidate user pair. Furthermore, we adapt these methods to address the top-k variant of the `STPSJoin` query, which returns only the top-k pairs of users with the highest similarity scores.

Building upon these methods, we proceed to address the problem of identifying spatio-textual user communities. We construct a user graph where edge weights represent spatio-textual user similarity scores. Then, we assign users to clusters by partitioning the graph in such a way that users belonging to the same cluster have higher spatio-textual similarity to each other than to users in other clusters. Partitioning is achieved using the Louvain method, a commonly used method for extracting communities from large networks [7]. This belongs to a family of algorithms that are based on the concept of *modularity*. Given a partitioning of a network, modularity provides a quantitative measure of its quality by measuring the density of edges inside communities to edges outside communities. The partitioning with the highest modularity score is the theoretically optimal one. Since examining all possible partitionings is practically infeasible, the Louvain method provides a greedy optimization algorithm for that task.

Applying the Louvain method requires as input a user graph where edge weights represent the importance of links between users. Unlike a social network graph, where these links are explicitly known (e.g., friends or followers), in our case the goal is to detect user communities based on spatio-textual similarities. Thus, to construct a complete and exact spatio-textual user graph, the spatio-textual similarities among all pairs of users have to be calculated, which is impractical for a large number of users. To overcome this problem, we propose a set of methods that reduce computation time by avoiding to compute the spatio-textual similarity of all pairs of users and/or the exact similarity scores. The main idea is to compute only a partial and approximate representation of the user similarity graph and apply the Louvain method on it. The goal is to reduce execution time, while the quality of the detected communities using this partial graph should still be as close as possible to the quality of those communities computed directly on the complete, exact graph.

The main 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, introducing also its top-k variant.
- We address the problem of spatio-textual user matching, through a series of algorithms for efficiently evaluating both the threshold-based and the top-k variant of the `STPSJoin` query.
- We introduce and formally define the problem of spatio-textual user clustering, modeling it as a community detection problem on a spatio-textual user similarity graph.
- We present a series of algorithms that significantly speed up the computation of user clusters relying on different approximations of the user graph, while also allowing a trade-off between the execution time and the quality of the identified clusters.
- Finally, we perform an extensive experimental evaluation using three large, real-world datasets. For the user matching problem, the results indicate that the proposed algorithms improve the execution time by an order of magnitude or more compared to a baseline method. For the clustering problem, the proposed techniques manage to significantly reduce the time required to identify user clusters, while achieving comparable results in terms of cluster quality.

This paper is an extension of our earlier work [18], where we have first introduced the `STPSJoin` query and the algorithms for its evaluation. Here, we build upon these methods and extend our approach to address the problem of spatio-textual user clustering.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 formally defines the user matching and clustering problems, while Sects. 4 and 5 present our algorithms. Section 6 presents the experimental evaluation of our proposed approaches. Section 7 concludes the paper.

2 Related work

2.1 Spatio-textual queries

To evaluate spatial keyword queries efficiently, current research enables the combination of spatial and textual indexes into hybrid spatio-textual structures. Established spatial indexes, including the R-tree, grid, and space-filling curves, have been combined with textual indexes, including inverted files and signature files. Characteristic examples are the indexes described in [51]. For instance, the R*-tree-IF is a hybrid structure where the top-level index comprises an R*-tree with inverted files attached to its leaf nodes. Conversely, in the IF-R*-tree the top-level index is an inverted file, in which the posting lists are indexed by an R*-tree. In a similar manner, the IR-tree is a hybrid index based on an R-tree, where each node is enhanced with a reference to an inverted file indexing the objects contained in the sub-tree rooted at that node [17,45]. Other combinations have also been investigated. For example, in [42], inverted files have been combined with a grid. A similar approach is followed in [13], but utilizing space-filling curves for spatial indexing. In [24], R-trees are combined with signature files, which are stored internally in the nodes of the tree. In [37], aR-Trees [34] are combined with inverted files. In the I^3 index [47], a quadtree is used for each individual keyword to spatially index the set of documents containing it. The RCA approach [48] uses only an inverted index, but maintains two inverted lists for each keyword. The first is a standard inverted list which stores the documents containing the keyword in decreasing order of relevance. The second stores documents according to the Z-order encoding of their coordinates.

Based on these, it is possible to characterize existing approaches as "text-first" or "space-first". A performance comparison along these lines is presented in [15]. Furthermore, a description of different variants of spatial keyword queries, including Boolean kNN queries, top-k kNN queries and Boolean range queries, and a comprehensive survey and experimental evaluation of existing algorithms can be found in [12].

Similarly to these works, our algorithms for spatio-textual point set join make use of hybrid spatio-textual index structures, in particular a spatial grid enhanced with inverted lists on its cells.

2.2 Similarity joins

Similarity joins return pairs of objects from given sets that satisfy a predefined similarity threshold. Next, we consider relevant types of joins.

Set similarity joins Set similarity joins identify pairs of sets, from two given collections, that have high similarity. This task is computationally challenging. A naive approach requires the consideration of the similarity between every possible pair of objects across sets. Due to their wide applications, set similarity joins have been extensively studied, especially with respect to textual characteristics, and a series of optimizations have been proposed. In [38], an inverted index-based probing method is proposed to reduce the number of potential candidates. In [11], it is observed that the prefixes of potential candidates must satisfy a minimal overlap. The ALL-PAIRS algorithm [5] further optimizes the size of the inverted index by visiting the tokens associated with every object with respect to their precomputed frequencies while also reducing the size of the indexing prefix by ordering the objects by their size. More recently, the proposed algorithms include the Adapt-Join [44] and PPJOIN+ [46]. The latter builds on ALL-PAIRS and introduces a positional filtering principle which exploits the ordering of tokens and operates on both the prefix and the suffix of the tokens of objects. Finally, an experimental analysis and evaluation on string similarity joins are presented in [28].

In our work, we use PPJOIN+ as an internal step of some of our algorithms, when needed to efficiently compute textual similarity joins.

Spatial joins Spatial joins return pairs of objects whose spatial distance is below a given threshold. Data structures and algorithms for spatial joins have been widely studied in the literature (see [27] for a survey). Both space partitioning and data partitioning structures have been applied for spatial joins. A commonly used algorithm is the one proposed in [9]. It traverses an R-tree starting from the root node and checking for pairs of child nodes with intersecting ϵ-extended minimum bounding rectangles (MBRs). An ϵ-extended rectangle is obtained by extending an MBR in every dimension by a given spatial threshold ϵ. Intersecting nodes are discovered following the plane sweeping method.

However, the problem of spatial joins over point sets has not received much attention. Some works have investigated similarity search for a collection of spatial point-set objects based on the Hausdorff distance [1,2]. The motivation behind that work is highly relevant to the STPSJoin query; however, there are important differences. We consider entities comprising sets of objects with both spatial and textual attributes, and we measure the distance among object 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 sets that are similar.

Spatio-textual joins Spatio-textual joins have attracted some attention recently. This process is primarily applied for duplicate detection, i.e., finding pairs of entities that are both spatially close and textually similar. The work in [4] is one of the first examples of spatio-textual join methods. It proposes the SpSJoin query, following the MapReduce paradigm for scalable computation of spatio-textual join queries. Several methods for spatio-textual similarity join using MapReduce have also been presented in [49]. Moreover, the spatio-textual join query has been studied in the form of spatial regions associated with textual descriptions [22,29,30]. Pruning strategies based on spatial and textual signatures of objects are employed to filter the number of candidates. Moreover, in [36], grid- and quadtree-based indexes are presented to efficiently partition the dataset either in a local or in a global fashion. This work also explores 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. The approach in [8] builds upon PPJ, a baseline method that extends PPJOIN+, to account for spatio-textual objects. 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 when presenting our algorithms.

Work on spatio-textual joins is highly relevant to our approach. However, the problem is different. Existing approaches deal with spatio-textual similarity joins among individual objects. On the contrary, we address spatio-textual similarity joins among object sets. The latter is required when objects are grouped with respect to a common characteristic. For example, in our case, this allows us to address the problem of matching and clustering users based on their sets of geotagged posts. Hence, the focus is on identifying similarities among groups of objects, rather than single objects.

2.3 Graph partitioning

A plethora of approaches exists for graph partitioning [6,10, 32,39], given the wide applicability of the problem, from road networks to VLSI design and parallel computing. Perhaps the most traditional and well-known family of methods is those based on *minimum cut* and its variations [39]. These methods aim at partitioning the graph into subgraphs in a way that the number of edges between partitions is minimized. These methods work well in applications where the number of partitions can be determined beforehand and it is desirable to generate partitions of roughly the same size (e.g., for load balancing).

Another family of algorithms, which are commonly used for community detection in graphs, is based on the measure of *modularity* [31]. Modularity measures the relative strength of links between nodes of the same partition as opposed to links between nodes of different partitions. Thus, maximizing the modularity score of the partitioning results in a better assignment of nodes into communities.

Since iterating over all possible node assignments to find the one that maximizes the value of modularity is practically infeasible for realistic size networks, heuristic algorithms are used. One of these methods, which is the one adopted in this paper, is the Louvain method [7]. It is a bottom-up approach. It first merges nodes into small communities by optimizing modularity locally. Then, it repeatedly merges smaller communities into larger ones hierarchically until no more improvement in the modularity score is observed (see

Sect. 5.1 for more details). Other methods also based on modularity optimization have been proposed in the literature [16,35,43]. In our work, we use the Louvain method as it has been shown to outperform other algorithms in this category with respect to both execution time and quality of identified communities [3,7].

More recently, some works have focused on community detection in networks where the nodes are associated with a geolocation [14,21,33]. These methods also adopt the modularity maximization approach, but take into consideration the factor of spatial proximity as well. This is done by using spatial distances in edge weights and/or combining spatial proximity with social links. However, none of these works addresses the problem of finding user groups based on a spatio-textual similarity measure. Finally, the work presented in [23] focuses on spatial-aware community search. The problem of community search differs from that of community detection; the latter partitions the entire graph into a set of communities, whereas the former is a query-dependent variant where the goal is to find the communities of a *given* vertex.

3 Problem definition

We define the `STPSJoin` query and then introduce the spatio-textual user matching and clustering problems.

3.1 Spatio-textual point-set join

Assume a collection $\mathcal{D}$ of spatio-textual objects (e.g., geotagged tweets or photographs) owned by a set of entities U (e.g., users). A spatio-textual object $o \in \mathcal{D}$ is a triple $o = \langle u, loc, doc \rangle$, where $u \in U$ is the owner of the object, $loc = \langle x, y \rangle$ is the object's location, and $doc = \langle \psi_1, \ldots, \psi_n \rangle$ is a set of keywords describing the object. We refer to the owner, location, and keyword set of 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 owned by entity u.

First, we define the spatio-textual similarity join between individual objects. This is based on two criteria, namely *spatial distance* and *textual similarity*. Similar to previous works [8,12], we define the spatial distance $\delta(o, o')$ between two objects as the Euclidean distance between their locations, and the textual similarity $\tau(o, o')$ as the Jaccard similarity of their keyword sets:

$$\tau\left(o, o'\right) = \frac{\left|o.doc \cap o'.doc\right|}{\left|o.doc \cup o'.doc\right|} \tag{1}$$

Given a spatial distance threshold ϵ_{loc} and a textual similarity 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}. Formally, the matching operator between objects is defined by the function μ:

$$\mu\left(o, o'\right) = \begin{cases} True, & if\, \delta\left(o, o'\right) \leq \epsilon_{loc} \& \tau\left(o, o'\right) \geq \epsilon_{doc} \\ False, & otherwise. \end{cases} \tag{2}$$

Notice that this definition corresponds to the spatio-textual similarity join condition used also in [8].

Next, we need to generalize this matching operator to *sets* of spatio-textual objects. The Jaccard similarity cannot be applied directly, as it measures the portion of *common* elements between two sets. Instead, we want to measure the portion of elements that *match* between two sets, according to the matching operator μ defined above. Thus, to define the similarity between two sets of spatio-textual objects, we adapt Jaccard similarity to use the spatio-textual matching operator μ as described below.

We extend function μ to account for the matching between an individual object o and a set of objects $D \subseteq \mathcal{D}$. In this case, we say that there is a match if o matches with at least one object in D. Formally:

$$\mu(o, D) = \begin{cases} True, & if\, \exists\, o' \in D \;:\; \mu\left(o, o'\right) = True \\ False, & otherwise. \end{cases} \tag{3}$$

Furthermore, let D and D' be two sets of spatio-textual objects. We use function $M(D, D')$ to define the set of objects in D that match with at least one object in D':

$$M\left(D, D'\right) = \{o \in D \;:\; \mu\left(o, D'\right) = True\} \tag{4}$$

We can now use M to define the spatio-textual similarity of two users.

Definition 1 (*Similarity*) Given two entities u, u' associated with the sets of objects D_u and $D_{u'}$, respectively, their spatio-textual similarity is defined as the fraction of the matched objects from one entity to the other divided by the total number of objects owned by the two entities, i.e.,

$$\sigma\left(u, u'\right) = \frac{|M\left(D_u, D_{u'}\right)| + |M\left(D_{u'}, D_u\right)|}{|D_u| + |D_{u'}|} \tag{5}$$

Note that $\sigma(u, u')$ is normalized in the interval $[0, 1]$ to account for the differences in the number of objects owned by different entities.

Based on the above, we proceed to define the *Spatio-Textual Point-Set Join* query (`STPSJoin`). `STPSJoin` identifies all pairs of object sets that have spatio-textual similarity higher than a specified threshold ϵ_u. To avoid returning duplicate pairs, we assume a total ordering over U, denoted by $\prec_U$. Formally, the `STPSJoin` query is defined as follows.

Definition 2 (`STPSJoin` *Query*) Given a collection $\mathcal{D}$ of spatio-textual objects belonging to a set of entities U, an `STPSJoin` query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, \epsilon_u \rangle$ returns a set $R \subseteq U \times U$ containing those pairs (u, u') such that $u \prec u'$ and $\sigma(u, u') \geq \epsilon_u$ with respect to the spatial and textual thresholds ϵ_{loc} and ϵ_{doc}.

3.2 Spatio-textual matching

Given the above, we can now formulate the problem of spatio-textual user matching. Assume a set of users U, where each user owns a set of geotagged posts, represented as spatio-textual objects, D_u. We introduce and address two variants of the problem, as presented formally below. The first identifies all pairs of matching users having spatio-textual similarity above a given threshold. The second is its top-k variant, i.e., it retrieves the top-k pairs of users in the dataset having the highest spatio-textual similarity score.

Problem 1.a (All-pairs User Matching) *Let $\mathcal{D}$ be a collection of spatio-textual objects belonging to a set of users U. The all-pairs spatio-textual user matching problem is to evaluate an* `STPSJoin` *query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, \epsilon_u \rangle$ over $\mathcal{D}$ and U.*

Problem 1.b (Top-k User Matching) *Let $\mathcal{D}$ be a collection of spatio-textual objects belonging to a set of users U. Given thresholds ϵ_{loc} and ϵ_{doc}, the problem of finding the top-k spatio-textual user pairs is to compute the k pairs of users with the highest similarity score, i.e., a set $R \subseteq U \times U$ such that $|R| = k$ and for any pair $(u_i, u_j) \in R$ it holds that $u_i \prec u_j$ and $\sigma(u_i, u_j) \geq \sigma(u_{i'}, u_{j'})$ for any pair $(u'_i, u'_j) \in U \times U \backslash R$.*

3.3 Spatio-textual clustering

Next, our goal is to identify clusters of similar users, where user similarity is defined according to the spatio-textual similarity measure introduced above. To that end, we assume a spatio-textual user similarity graph, where edges connect pairs of users having a nonzero spatio-textual similarity score. Formally, this spatio-textual user similarity graph is defined as follows.

Definition 3 (*Similarity Graph*) Given a collection $\mathcal{D}$ of spatio-textual objects belonging to a set of users U, and thresholds $\epsilon_{loc}, \epsilon_{doc}$, the spatio-textual user similarity graph is an undirected, weighted graph $G = (V, E, W)$, where $V = U$ and $E = \{(u, u') \in U \times U : \sigma(u, u') > 0\}$, i.e., E contains those user pairs that belong to the result set of an `STPSJoin` query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, 0 \rangle$. Each edge $e_{u,u'} \in E$ is assigned a weight $W(u, u') = \sigma(u, u')$.

Given the above graph G, computing user clusters corresponds to computing a partitioning of G. The goal is to identify a set of partitions such that the similarities between users within the same cluster are higher than the similarities between users in different clusters. A formal measure of partition quality is provided by the concept of *modularity* [31]. This is a commonly used criterion for community detection, which has been applied in several domains, from networks on the Web to networks in biology. Given a partitioning of a network, modularity is a scalar value between −1 and 1 that measures the density of links inside partitions as opposed to links across partitions. Higher values indicate more cohesive partitions.

Following [16], the modularity of a partitioning P of a weighted graph G is formally defined as:

$$mod(G, P) = \frac{1}{2\, w_{total}} \sum_{u,u'} \left[w_{u,u'} - \frac{w_u\, w_{u'}}{2\, w_{total}} \right] \delta\left(c_u, c_{u'}\right) \tag{6}$$

where

- $w_{u,u'}$ is the weight of the edge between vertices u and u';
- $w_u = \sum_{u'} w_{u,u'}$ is the weighted degree of vertex u, i.e., the aggregate similarity between u and every other adjacent vertex u';
- c_u denotes the cluster to which u is assigned according to the partitioning P;
- $\delta(c_u, c_{u'})$ is a function that returns 1 if $c_u = c_{u'}$, and 0 otherwise;
- $w_{total} = \frac{1}{2} \sum_{u,u'} w_{u,u'}$ is the sum of all the weights in the network.

Based on the above, we can formally state the spatio-textual user clustering problem as follows.

Problem 2 (User Clustering) *Given a collection $\mathcal{D}$ of spatio-textual objects belonging to a set of users U, and thresholds ϵ_{loc}, ϵ_{doc}, the spatio-textual user clustering problem is to compute a partitioning P of the corresponding spatio-textual user similarity graph G that maximizes the modularity $mod(G, P)$.*

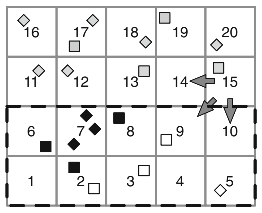
4 Spatio-textual user matching

We first address the all-pairs variant and then adapt our methods to the top-k problem.

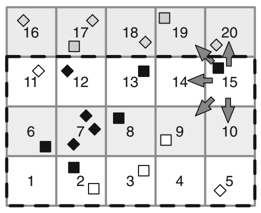
4.1 All-pairs algorithms

4.1.1 Algorithm `S-PPJ-C`

Our goal is to find all pairs of users whose spatio-textual similarity score σ exceeds a given threshold ϵ_u. A straightforward



(a) PPJ-C traversal.



(b) PPJ-B traversal.

Fig. 2 The grid traversal strategies of PPJ-C (**a**) and PPJ-B (**b**)

algorithm is to compute, for each pair of users (u, u'), their sets of matching objects, $M(D_u, D_{u'})$ and $M(D_{u'}, D_u)$, and then check whether the resulting similarity score σ exceeds ϵ_u. For each user pair (u, u'), the problem is cast as a spatio-textual similarity join query, ST-SJOIN$(D, \epsilon_{loc}, \epsilon_{doc})$, where $D = D_u \cup D_{u'}$ [8]. This query returns all pairs of objects $(o, o') \in D \times D$ such that $\delta(o, o') \le \epsilon_{loc}$ and $\tau(o, o') \ge \epsilon_{doc}$. From this result, we can derive the sets $M(D_u, D_{u'})$ and $M(D_{u'}, D_u)$ and, subsequently, the similarity score σ.

To efficiently compute ST-SJOIN$(D, \epsilon_{loc}, \epsilon_{doc})$, the PPJ-C algorithm [8] can be used. PPJ-C partitions the space by constructing a grid at query time, with cells having an extent in each dimension equal to the spatial distance threshold ϵ_{loc}. Cell ids are assigned bottom-up, in a row-wise order. Objects in each visited cell c need to be joined only with those in c and in the adjacent cells with ids lower than c. Thus, for each cell, one self-join and four non-self-joins are executed. These are performed using the PPJ algorithm, that extends the set similarity join algorithm PPJOIN [46] by including an additional check on the spatial distance.

Figure 2a illustrates an example. A spatial grid is constructed over a set of objects of two different users, u (depicted as squares) and u' (depicted as diamonds). The algorithm visits the cells sequentially according to the displayed ids. When cell c_{15} is reached, join operations are executed with the objects in cells c_9, c_{10}, c_{14} and c_{15}. Objects that have been found to match so far are displayed in black, whereas those that do not match are displayed in white. Objects for which the search has not yet concluded are colored in gray.

Using PPJ-C as basis, we can derive a baseline algorithm for our problem, denoted as S-PPJ-C (Set-PPJ-C). In our case, the input to each execution of PPJ-C is the union of the object sets of two users, $D_u \cup D_{u'}$. We maintain the following additional information during the construction of the grid:

1. For each cell c, we maintain in separate lists the objects belonging to each user. We denote by D_u^c the set of objects of user u that are contained in c.
2. For each user u, we maintain a list of cells C_u that contain objects belonging to u. Each list C_u is sorted according to cell ids in ascending order.

Algorithm 1: S-PPJ-C

```
   Input: $D, U, \epsilon_{loc}, \epsilon_{doc}, \epsilon_u$
   Output: Pairs of matched users $R$
1  $R \leftarrow \emptyset$
2  $selectedUsers \leftarrow \emptyset$
3  $G \leftarrow createGridIndex(D, U, \epsilon_{loc})$
4  foreach $u_j \in U$ do
5      foreach $u_i \in selectedUsers$ do
6          $M(D_{u_i}, D_{u_j}), M(D_{u_j}, D_{u_i}) \leftarrow$
           $PPJ\text{–}C(D_{u_i} \cup D_{u_j}, \epsilon_{loc}, \epsilon_{doc})$
7          $\sigma \leftarrow$
           $(|M(D_{u_i}, D_{u_j})| + |M(D_{u_j}, D_{u_i})|) \ / \ (|D_{u_i}| + |D_{u_j}|)$
8          if $\sigma \ge \epsilon_u$ then
9              $R.add(\langle u_i, u_j \rangle)$
10     $selectedUsers.add(u_j)$
11 return $R$
```

S-PPJ-C is presented in Algorithm 1. It loops through all pairs of users (lines 4–5), taking into consideration the total ordering $\prec_U$ of the user set U. For each pair, it executes an adapted, non-self-join version of the PPJ-C algorithm (line 6), matching only pairs of objects belonging to different users, based on the lists C_{u_i} and C_{u_j}. The results of PPJ-C are used to compute the similarity score σ for that user pair and to check whether it exceeds the given threshold, in which case it is added to the result (lines 7–9).

4.1.2 Algorithm S-PPJ-B

S-PPJ-C performs several unnecessary computations, since for each pair of users it needs to find all their matching objects. Next, we describe a more efficient algorithm, S-PPJ-B (B stands for bound), which stops the computation as soon as it can determine that the similarity is below the required threshold ϵ_u.

S-PPJ-B replaces the execution of PPJ-C (line 6 in Algorithm 1) with a modified process, denoted as PPJ-B, as described next. PPJ-B uses an upper bound on the allowed number of unmatched objects to prune the search on the grid. While examining two users, it uses the user similarity threshold ϵ_u and the number of objects belonging to each user to compute an upper bound on the number of allowed unmatched objects between the two users, above which the user similarity cannot exceed ϵ_u.

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.,

$$\beta_{u,u'} = |D_u| + |D_{u'}| - |M(D_u, D_{u'})| - |M(D_{u'}, D_u)| \quad (7)$$

We compute an upper bound for $\beta_{u,u'}$ as indicated by the following lemma.

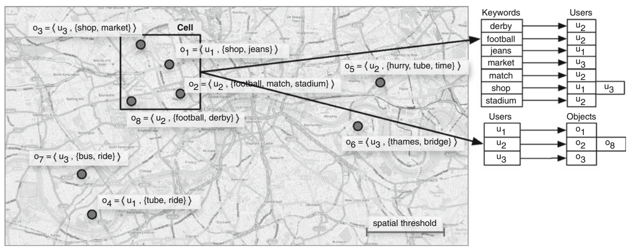

$o_3 = \langle u_3, \{shop, market\} \rangle$
Cell
$o_1 = \langle u_1, \{shop, jeans\} \rangle$
$o_5 = \langle u_2, \{hurry, tube, time\} \rangle$
$o_2 = \langle u_2, \{football, match, stadium\} \rangle$
$o_8 = \langle u_2, \{football, derby\} \rangle$
$o_6 = \langle u_3, \{thames, bridge\} \rangle$
$o_7 = \langle u_3, \{bus, ride\} \rangle$
$o_4 = \langle u_1, \{tube, ride\} \rangle$
spatial threshold
Keywords
Users
derby
football
jeans
market
match
shop
stadium
u_2
u_2
u_1
u_3
u_2
u_1
u_3
u_2
Users
Objects
u_1
u_2
u_3
o_1
o_2
o_8
o_3

Fig. 3 Spatio-textual index structure used by `S-PPJ-F`

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

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

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

□

To exploit this bound more efficiently, `PPJ-B` employs a modified grid traversal strategy. While traversing each row bottom-up, it treats odd and even rows differently (considering the bottom row as 1). If a cell $c_{i,j}$ belongs to an odd row, the contained objects are matched with objects from all surrounding cells, except the cell directly on the right. Otherwise, objects are matched only with the cell that is on the left. With this modification, as illustrated in Fig. 2, when cell c_{15} is reached, `PPJ-B` has determined the state of all objects in cells up to c_{15}, while `PPJ-C` only those up to cell c_{10}. Once `PPJ-B` examines the last cell of an odd row, it has considered every potential match for any object it has encountered up to that point. Then, it checks whether the number of unmatched objects exceeds the bound β, and, if so, the search stops.

4.1.3 Algorithm `S-PPJ-F`

`S-PPJ-B` still needs to iterate over all pairs of users. Next, we present the `S-PPJ-F` algorithm, which further increases efficiency by following a filter and refine strategy. `S-PPJ-F` uses a spatio-textual index structure constructed at runtime, as illustrated in Fig. 3. It is a dynamic grid enhanced with two inverted indexes in each cell. The first maps keywords to users having objects with that keyword in the cell. The second maps users to their objects located in that cell.

Algorithm 2: `S-PPJ-F`

```
   Input: $D$, $U$, $\epsilon_{loc}$, $\epsilon_{doc}$, $\epsilon_u$
   Output: Pairs of matched users $R$
1  $R \leftarrow \emptyset$
2  $G \leftarrow initialiseSTGridIndex(D, \epsilon_{loc})$
3  foreach $u \in U$ do
4      $\mathcal{C}_u[u'], \mathcal{C}_{u'}[u] \leftarrow$ Filter($u, G$)
5      foreach $u' \in \mathcal{C}_u.keys()$ do
6          Refine($u, u'$)
7      $G.insert(D_u)$
8  return $R$
9
10 Procedure Filter($u, G$)
11     foreach $c \in C_u$ do
12         $T \leftarrow getKeywords(u, c)$
13         foreach $c' \in G.getRelevantCells(c)$ do
14             foreach $t \in T$ do
15                 foreach $u' \in G.getUsersWithKeyword(c', t)$
                   do
16                     $\mathcal{C}_u[u'].add(c), \mathcal{C}_{u'}[u].add(c')$
17     return $\mathcal{C}_u[u'], \mathcal{C}_{u'}[u]$
18
19 Procedure Refine($u, u'$)
20     $\bar{\sigma} \leftarrow computeBound(u, u')$
21     if $\bar{\sigma} \geq \epsilon_u$ then
22         $\sigma \leftarrow PPJ\text{–}B(D_u \cup D_{u'}, \epsilon_{loc}, \epsilon_{doc}, \epsilon_u)$
23         if $\sigma \geq \epsilon_u$ then
24             $R.add(\langle u', u \rangle)$
```

`S-PPJ-F` is outlined in Algorithm 2. First, the grid is constructed, with the inverted indexes in the cells being empty (line 2). Then, the algorithm iterates over the users (line 3). For each visited user u, it first executes a filtering step (line 4) to identify potential matches among the previously seen users. For each of those, a refinement step is executed (line 6) to determine whether there is an actual match. Finally, the objects of u are inserted in the index (line 7), so that u itself is considered as a candidate in the subsequent iterations. The filtering and refinement steps are detailed below.

When a user u is selected, the algorithm examines each cell $c \in C_u$ and retrieves the set of keywords T appearing in the respective objects (lines 11–12). This is utilized to populate a list of candidate matching users extracted from c and its relevant adjacent cells (lines 13–16). These are users with objects that appear in one of these cells and contain at least one keyword from T. For each candidate matching user u', two lists are maintained: $\mathcal{C}_u[u']$, holding those cells that contain objects of u that potentially match (both spatially and textually) with objects of u', and, $\mathcal{C}_{u'}[u]$, vice versa (line 16). Then, for each user u', an upper bound $\bar{\sigma}$ of the similarity score between u and u' is calculated (line 20), assuming that all of their objects which are contained in the same or adjacent cells match:

$$\bar{\sigma}\left(u, u'\right) = \frac{\sum_{c \in \mathcal{C}_u[u']} \left|D_u^c\right| + \sum_{c' \in \mathcal{C}_{u'}[u]} \left|D_{u'}^{c'}\right|}{|D_u| + |D_{u'}|} \tag{8}$$

If $\bar{\sigma}(u, u') < \epsilon_u$, then this pair can be safely pruned. Otherwise, a refinement step follows, where PPJ-B is executed to calculate the exact similarity score and accordingly add this pair to the results (lines 21–24).

4.2 Top-*k* algorithms

4.2.1 Algorithm TOPK-S-PPJ-F

TOPK-S-PPJ-F is a direct adaption of S-PPJ-F. It is outlined in Algorithm 3. The main modifications 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 found. The user similarity threshold ϵ_u is set to the similarity score of the kth best pair in the queue and is updated accordingly. It is then used in the filtering phase in a similar manner to S-PPJ-F. TOPK-S-PPJ-F examines users in ascending order of the size of their object sets (line 4). The reason for this is that users with larger object sets require more computations than users with fewer objects. Thus, by the time the algorithm reaches the most computationally demanding users, a higher user similarity threshold may have been computed.

Algorithm 3: TOPK-S-PPJ-F

```
   Input: $D, U, \epsilon_{loc}, \epsilon_{doc}, k$
   Output: Top-$k$ user pairs $R$
1  $R \leftarrow \emptyset$
2  $G \leftarrow initialiseSTGridIndex(D, \epsilon_{loc})$
3  $\epsilon_u \leftarrow -1$
4  foreach $u \in sorted(U)$ do
5      $\mathcal{C}_u[u'], \mathcal{C}_{u'}[u] \leftarrow$ Filter$(u, G)$    // same as in Algorithm 2
6      foreach $u' \in \mathcal{C}_u.keys()$ do
7          Refine$(u, u')$
8      $G.insert(D_u)$
9  return $R$
10
11 Procedure Refine$(u, u')$
12     $\bar{\sigma} \leftarrow computeBound(u, u')$
13     if $\bar{\sigma} \geq \epsilon_u$ then
14         $\sigma \leftarrow PPJ\text{–}B(D_u \cup D_{u'}, \epsilon_{loc}, \epsilon_{doc}, \epsilon_u)$
15         if $\sigma > \epsilon_u$ then
16             $R.update(\langle u', u\rangle)$
17             if $|R| = k$ then
18                 $\epsilon_u \leftarrow R.get(k).score$
```

4.2.2 Algorithm TOPK-S-PPJ-S

TOPK-S-PPJ-S operates similarly to TOPK-S-PPJ-F, but uses a heuristic strategy to decide the order in which users are evaluated. User objects are placed in a spatial grid. Then, each cell c is assigned a score indicating how many users have objects located in c or in its adjacent cells. Users are then assigned a score by summing, for every object o belonging to them, the score of the cell that o is located in. This prioritizes users with objects in "popular" cells, as they are more likely to match with other users. The goal is to identify high scoring pairs more quickly. The score of a cell c is calculated as:

$$score(c) = \left|\cup_{c' \in G.getRelevantCells(c)} G.getUsers\left(c'\right)\right| \tag{9}$$

where G is the spatial grid, c is a cell in the grid, $G.getUsers(c)$ 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 as follows:

$$score(u) = \sum_{o \in D_u} score(o_c) \tag{10}$$

where o_c denotes the cell that object o is located in.

4.2.3 Algorithm TOPK-S-PPJ-P

TOPK-S-PPJ-P introduces an additional filtering step. Similarly to TOPK-S-PPJ-F, users are visited in ascending order of the size of their object sets. For every user u, the algorithm computes an upper bound on the similarity score between u and any other user u' that has been previously examined. This is done by identifying the objects in D_u that match with any object in $D_{U'}$, where the latter is the union of the objects of all users examined up to that point. We denote this set as $M_{u,U'}$. This computes an upper bound on $\sigma(u, u')$ for every user u' that has been visited prior to u:

$$\bar{\sigma}_u = \frac{\left|M_{u,U'}\right| + \max_{u' \in U'} |D_{u'}|}{|D_u| + \max_{u' \in U'} |D_{u'}|} \tag{11}$$

The following lemma shows that if the users are visited in ascending order of the size of their object sets, then $\bar{\sigma}_u$ is an upper bound on the similarity score between u and any user u' encountered before it.

Lemma 2 *Assume a user u and a set of users U', such that $|D_{u'}| \leq |D_u|$ for any $u' \in U'$. Then, it holds that $\sigma(u, u') \leq \bar{\sigma}_u$.*

Proof Let $m_u = |M_{u,U'}|$, $m_{u,u'} = |M(D_u, D_{u'})|$, $d_u = |D_u|$ and $d_{max} = \max_{u' \in U'} |D_{u'}|$. Then, since $m_u \geq m_{u,u'}$ and $d_u \geq m_u$, it holds that:

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

Thus, it suffices to show that:

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

Indeed:

$$\begin{aligned}
&\frac{m_u + d_{max}}{d_u + d_{max}} < \frac{m_u + d_{u'}}{d_u + d_{u'}} \\
&\quad \Rightarrow m_u \cdot d_u + m_u \cdot d_{u'} + d_{max} \cdot d_u + d_{max} \cdot d_{u'} \\
&\qquad < m_u \cdot d_u + m_u \cdot d_{max} + d_{u'} \cdot d_u + d_{u'} \cdot d_{max} \\
&\quad \Rightarrow m_u \cdot d_{u'} + d_{max} \cdot d_u < m_u \cdot d_{max} + d_{u'} \cdot d_u \\
&\quad \Rightarrow (d_{max} - d_{u'}) \cdot d_u < (d_{max} - d_{u'}) \cdot m_u \\
&\quad \Rightarrow d_u < m_u
\end{aligned}$$

which is false. □

To avoid computing exact similarity scores among objects and to speed up the bound calculation, we utilize again the spatio-textual index shown in Fig. 3 and we place in $M_{u,U'}$ all objects with a keyword that appears (due to a previously selected user) in the same or adjacent cell. This allows 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.

5 Spatio-textual user clustering

In the following, we extend our approach to address the spatio-textual clustering problem. As defined in Sect. 3.3, our aim is to compute a partitioning of the spatio-textual user similarity graph. The quality of a given partitioning is measured by the modularity score (see Eq. 6). However, examining all possible network partitionings is clearly infeasible for realistic size graphs. Nevertheless, several greedy optimization algorithms have been studied in the literature, which employ heuristics to significantly speed up computation time, while still discovering partitions with high modularity value. Next, we present some preliminaries, outlining how the Louvain method [7] works, which is a commonly used algorithm for this purpose. Then, we introduce our approach and algorithms, which combine the Louvain method and our previously presented algorithms for computing spatio-textual similarities between users.

5.1 Preliminaries

The Louvain method is a greedy algorithm for community detection in graphs based on the concept of modularity optimization. It comprises two stages, which are applied repeatedly.

During the first stage, the algorithm iterates over the nodes of the graph. Initially, each node is assigned to its own community. For each node, the algorithm calculates the difference in modularity that would result by moving this node from its current community to any of the communities of its adjacent nodes. This difference in modularity by reassigning u to another community c can be calculated as follows:

$$\begin{aligned}
\Delta mod(u, c) = &\left[\frac{w_{in}^c + w_u^c}{2w_{total}} - \left(\frac{w^c + w_u}{2w_{total}}\right)^2\right] \\
&- \left[\frac{w_{in}^c}{2w_{total}} - \left(\frac{w^c}{2w_{total}}\right)^2 - \left(\frac{w_u}{2w_{total}}\right)^2\right]
\end{aligned} \tag{12}$$

where w_{in}^c is the sum of the weights of the edges inside c, w^c is the sum of the weights of the edges adjacent to the nodes in c, and w_u^c is the sum of the weights of the edges from u to nodes in c.

If any of these candidate reassignments has a positive effect, i.e., $\Delta mod(u, c) > 0$, the one achieving the highest increase is executed. Otherwise, the node remains in its current community. The process is repeated until a complete pass over all nodes ends without any reassignments occurring.

In the second stage, the resulting communities are used to construct a new, higher-level graph. This is derived by aggregating nodes of the same community into a single node, aggregating edges between communities into edges between the new nodes, and aggregating edges inside communities into self-loops. Edge weights in this new graph are calculated by the sum of the weights of the corresponding edges from which they are derived. Then, the first stage is executed using this graph as input.

The process is repeated, until an execution of the first stage completes without having performed any reassignments.

5.2 Approach

A straightforward solution to the spatio-textual user clustering problem is to combine the `STPSJoin` query and the Louvain method in a two-phase process: the first phase computes the spatio-textual user similarity graph, while the second computes a partitioning of this graph. This provides a generic process for solving the spatio-textual clustering problem as outlined in Algorithm 4.

Specifically, in the first phase, which comprises the function `ComputeUserGraph()`, an `STPSJoin` query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, 0 \rangle$ is executed over the set of users U and objects D to compute the spatio-textual user similarity graph. In what follows, we refer to this graph as the *exact graph* G. In the second phase, which comprises the function `ComputeClusters()`, the Louvain method is executed on G to compute a partitioning P that assigns users to clusters.

Due to its greedy heuristic, the Louvain method can achieve relatively low execution time even for quite large graphs. Thus, the execution time of the whole process is dominated by the time needed to compute the `STPSJoin` query for generating the exact graph G. This is true even when exploiting our more efficient algorithms for this purpose (in particular, `S-PPJ-F`), which reduce execution time by one or more orders of magnitude. Indeed, our experiments showed that, in the two largest considered datasets, the time spent by the Louvain method was less than 1% of the total execution time. Consequently, the computation of the exact graph G constitutes the bottleneck in the whole process.

To overcome this problem, the main idea of our approach is to replace the computation of the exact graph G with that of an *approximate graph* G^*, on which then the partitioning can be performed. For that purpose, this approximate graph G^* should have the following properties:

– computing G^* should be *much faster* than computing G; and
– the user clusters obtained via G^* should have *comparable quality* to those obtained via G.

The latter is measured as follows. Let P denote the partitioning computed on G, whereas P' the one computed on G^*. For both partitionings, we calculate their modularity when applied on G, i.e., $mod(G, P)$ and $mod(G, P')$, respectively. Then, an approximate graph G^* is effective, if it results in a $mod(G, P')$ that is close to $mod(G, P)$.

Moreover, a desirable property is that the method allows a trade-off between execution time and quality of results, by controlling how coarse-grained or fine-grained the approximate graph G^* is with respect to the exact graph G.

Motivated by these, in the next section we present several methods that elaborate on the steps of the generic process to derive more efficient algorithms for solving the problem.

Algorithm 4: Generic Process for the Spatio-Textual User Clustering Problem

```
  Input: $D$, $U$, $\epsilon_{loc}$, $\epsilon_{doc}$
  Output: Partitioning $P$ assigning users to clusters
1 $G \leftarrow$ ComputeUserGraph($D, U, \epsilon_{loc}, \epsilon_{doc}$)
2 $P \leftarrow$ ComputeClusters($G$)
3 return $P$
```

5.3 Algorithms

As explained above, the main idea for speeding up the computation of the user clusters is to compute an approximate graph G^* instead of the exact graph G. In the exact case, the `ComputeUserGraph()` function (line 1 in Algorithm 4) executes an `STPSJoin` query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, 0 \rangle$ over U and D, returning all matching pairs of users and their exact similarity scores. Based on these, it builds the exact user similarity graph $G = (U, E_G, W_G)$, where:

– E_G contains the pairs returned by the `STPSJoin` query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, 0 \rangle$
– $W_G(u, u') = \sigma(u, u')$.

Accordingly, producing an approximate graph G^* can be accomplished by:

– partially computing the set of edges of the graph; or
– partially computing (i.e., approximating) the edge weights.

Following that, we present below two sets of methods for this purpose.

5.3.1 Methods based on partial edge computation

The first two presented strategies rely on computing only a selection of top-k edges of the graph, either globally (i.e., on the whole graph), or locally (i.e., per user). In both cases, the trade-off between computation time and quality of results is determined by the respective parameter k, which controls the number of edges that are computed.

Top-k user similarity graph (G^*-`TOPK`) Our first strategy is motivated by the fact that, to identify clusters, we are primarily interested in the strongest connections among users, i.e., those edges having the highest similarity scores. This can be done by computing the top-k spatio-textual user pairs.

In this case, the `ComputeUserGraph()` function executes any of the algorithms presented in Sect. 4.2 for this purpose and uses the returned results to produce the approximate graph G^*. We refer to this variant of the approximate graph as G*-TOPK. It is a graph that contains only k edges, with the exact weight set on each edge.

More specifically, G*-TOPK $= (U, E_{\text{TOPK}}, W_{\text{TOPK}})$, where:

- E_{TOPK} contains the pairs corresponding to the top-k results of the STPSJoin query $Q = \langle \epsilon_{loc}, \epsilon_{doc}, 0 \rangle$
- $W_{\text{TOPK}}(u, u') = \sigma(u, u')$.

Hence, $E_{\text{TOPK}} \subseteq E_{\text{G}}$ and $W_{\text{TOPK}}(u, u') = W_{\text{G}}(u, u')$.

Afterward, the ComputeClusters() function executes the Louvain method on G*-TOPK to assign users to clusters.

A drawback of this strategy is that many users may not appear in any of the top-k edges. These users will have no connections to other users in the G*-TOPK graph; hence, they will be left out of the produced clusters. A workaround is to assign these users to clusters by explicitly computing their similarities to other users in those clusters. However, the overhead of such an additional step eventually negates the previous gains in execution time. Moreover, some connections may be important locally but not globally and hence will be missed. Finally, since the total number of edges is not known (given that avoiding performing a full join is exactly the purpose here), it is not straightforward how to select a reasonable value for k.

kNN user similarity graph (G*-KNN) Our second strategy attempts to address the aforementioned drawbacks. It has a similar motivation to the previous one, i.e., it is also biased toward computing edges with high similarity scores. However, instead of identifying the most similar pairs of users overall, it identifies for each user its k-nearest neighbors. We refer to the resulting graph in this case as G*-KNN. In this graph, each user u will have (at most) k edges, linking it to its k-nearest neighbors. The weights on these edges are exact similarity scores.

Formally, G*-KNN $= (U, E_{\text{KNN}}, W_{\text{KNN}})$, where:

- E_{KNN} contains the pairs corresponding to the results of executing a kNN query for each user
- $W_{\text{KNN}}(u, u') = \sigma(u, u')$.

Hence, $E_{\text{KNN}} \subseteq E_{\text{G}}$ and $W_{\text{KNN}}(u, u') = W_{\text{G}}(u, u')$.

Next, we describe an algorithm, called KNN-S-PPJ-F, that can be used to efficiently execute a kNN query for each user. The process is outlined in Algorithm 5 and is an adaptation of the TOPK-S-PPJ-F algorithm presented in Algorithm 3. Contrary to TOPK-S-PPJ-F, at the beginning of the process all users are inserted in the spatio-textual grid index (lines 3–4). The k-nearest neighbors for each user are then identified in a fashion similar to TOPK-S-PPJ-F. There are two main differences. First, the priority queue now holds pairs for the user evaluated in the current loop. Upon termination of the process for this user, the results are appended to the overall results (line 13) and the priority queue is cleared (line 6). Second, the map J is used to maintain the results of executed exact joins (line 21). This is useful in order to avoid the duplicate execution of the join operation for pairs that have already been evaluated in previous iterations (line 23).

Algorithm 5: KNN-S-PPJ-F

```
    Input: $D, U, \epsilon_{loc}, \epsilon_{doc}, k$
    Output: A map $R$ containing, for each user, its $k$-nearest
            neighbors
1   $G \leftarrow initialiseSTGridIndex(D, \epsilon_{loc})$
2   $J \leftarrow \emptyset$
3   foreach $u \in (U)$ do
4       $G.insert(D_u)$
5   foreach $u \in U$ do
6       $R_u \leftarrow \emptyset$
7       $\epsilon_u \leftarrow -1$
8       $G.remove(D_u)$
9       $\mathcal{C}_u, \mathcal{C}_{u'} \leftarrow$ Filter$(u, G)$
10      foreach $u' \in \mathcal{C}_u.keys()$ do
11          Refine$(u, u')$
12      $G.insert(D_u)$
13      $R.addAll(R_u)$
14  return $R$
15
16  Procedure Refine$(u, u')$
17      $\bar{\sigma} \leftarrow computeBound(u, u')$
18      if $\bar{\sigma} \geq \epsilon_u$ then
19          if $\langle u', u \rangle \notin J.keys()$ then
20              $\sigma \leftarrow PPJ\text{–}B(D_u \cup D_{u'}, \epsilon_{loc}, \epsilon_{doc}, \epsilon_u)$
21              $J[\langle u', u \rangle] \leftarrow \sigma$
22          else
23              $\sigma \leftarrow J[\langle u', u \rangle]$
24          if $\sigma > \epsilon_u$ then
25              $R_u.update(\langle u', u \rangle)$
26              if $|R_u| = k$ then
27                  $\epsilon_u \leftarrow R_u.get(k).score$
```

Thus, in this method, the ComputeUserGraph() function invokes the KNN-S-PPJ-F process to produce the approximate graph G*-KNN. Afterward, the function ComputeClusters() executes the Louvain method on G*-KNN to identify user clusters.

5.3.2 Methods based on partial edge weights computation

In the following, we present three methods that rely on a different rationale, in particular on computing approximate values for the user similarities. The idea in this case is to reduce execution time during the calculation of edge weights, by replacing the computation of exact similarity scores with respective bounds, which can be upper, lower, or combination of both.

These methods involve two main operations. The first one concerns the construction of the approximate graph and is performed by the `ComputeUserGraph()` function. The second concerns the refinement of (some of) the edge weights in order to improve the accuracy of the results. Thus, it provides a means to adjust the balance between execution time and quality of results. This operation takes place during execution of the function `ComputeClusters()`.

First, we describe how the approximate graph is constructed in each of the three methods.

Upper-bound user similarity graph (G*-UB) We have already seen in the S-PPJ-F algorithm (Sect. 4.1.3), how a spatio-textual index can be exploited to efficiently compute an upper bound $\bar{\sigma}(u, u')$ on the similarity between two users u and u'. Here, we leverage this to derive an approximate graph, called G*-UB, where edge weights are upper bounds of similarity scores rather than exact ones.

Specifically, G*-UB = $(U, E_{\text{UB}}, W_{\text{UB}})$, where:

- E_{UB} contains those pairs for which $\bar{\sigma}(u, u') > 0$
- $W_{\text{UB}}(u, u') = \bar{\sigma}(u, u')$.

Hence, $E_{\text{UB}} \supseteq E_{\text{G}}$ and $W_{\text{UB}}(u, u') \geq W_{\text{G}}(u, u')$.

In this method, the function `ComputeUserGraph()` executes a process similar to the S-PPJ-F algorithm (see Algorithm 2). The only difference is that, in this case, the procedure `Refine()` returns as soon as the function `computeBound()` finishes (line 20), without invoking the much more costly process of PPJ-B.

Lower-bound user similarity graph (G*-LB) This method works analogously to the one presented above, but instead it computes a lower bound on the similarity score between two users.

We first describe how this lower bound is computed. The process is again similar to the S-PPJ-F algorithm, utilizing its spatio-textual index. For each user u, we search for candidate matches exclusively in the cells which are occupied by this user's objects. We filter candidate pairs by selecting every other user u' having at least one object in one of these cells with a token that also appears in one of the objects of u from the same cell. For each pair of users that qualifies this filtering step, we execute a modified version of PPJ-B. This applies a textual join operation for the objects in cells shared by both users. Although a join operation is performed in this case, the computation is much faster since it is limited only to the objects contained in the same cell instead of considering all adjacent cells too.

The result of this step is the calculation of the lower bound of the similarity of the two users, since objects are considered to be matching only if they occupy exactly the same cell. This lower bound $\hat{\sigma}(u, u')$ of the similarity between two users u and u' is formally defined as:

$$\hat{\sigma} = \frac{\forall_{c \in C_u \cap C_{u'}} \left|M\left(D_u^c, D_{u'}^c\right)\right| + \left|M\left(D_{u'}^c, D_u^c\right)\right|}{|D_u| + |D_{u'}|} \tag{13}$$

Based on the results, we derive a graph G*-LB = $(U, E_{\text{LB}}, W_{\text{LB}})$, where:

- E_{LB} contains those pairs for which $\hat{\sigma}(u, u') > 0$
- $W_{\text{LB}}(u, u') = \hat{\sigma}(u, u')$.

Hence, $E_{\text{LB}} \subseteq E_{\text{G}}$ and $W_{\text{LB}}(u, u') \leq W_{\text{G}}(u, u')$.

Therefore, here the function `ComputeUserGraph()` computes the graph G*-LB by calculating lower bounds on the similarity between users as explained above.

Upper–lower-bound user similarity graph (G*-ULB) Both methods relying on G*-UB and G*-LB reduce computation time by avoiding the costly operations for computing exact edge weights. However, both of them also have some drawbacks. In the former, the set of edges that are produced contains false positives, since there may exist a potentially large number of user pairs (u, u') for which $\bar{\sigma}(u, u') > 0$ but $\sigma(u, u') = 0$. This also implies an additional overhead for the execution of the Louvain method. Inversely, in the latter case, the set of edges contains false negatives, since there may exist many user pairs (u, u') for which $\hat{\sigma}(u, u') = 0$ but $\sigma(u, u') > 0$. This means that the generated graph lacks certain links, which may be important for identifying more accurate clusters. An additional drawback of the upper-bound-based method is the following. The way the upper bound is computed is rather optimistic, since it assumes that all the objects located in the same or in any of the adjacent cells match. Thus, in many cases, this upper bound is not very tight. Accordingly, using this value as edge weight instead of the exact similarity score implies that in many cases the weight of the link among two users may be largely overestimated. This, in turn, leads to wrong decisions when reassigning users to clusters during the first stage of the Louvain method.

Consequently, our final strategy attempts to address the above shortcomings by using both upper and lower bounds for producing the approximate graph G^*. The procedures for computing each type of bound, as described above, are combined and performed together in a single pass. In fact, a simplifying modification can be made because, as shown below, the process only needs to identify the pairs of users (u, u') having $\bar{\sigma}(u, u') > 0$, without computing the actual value of $\bar{\sigma}(u, u')$. This process computes the lower bound of the similarity scores among candidate user pairs, but it also returns all user pairs having an upper similarity bound greater than zero.

As a result, we construct a graph G*-ULB = (U, E_{ULB}, W_{ULB}), where:

- E_{ULB} contains those pairs for which $\bar{\sigma}(u, u') > 0$
- $W_{\text{ULB}}(u, u') = \hat{\sigma}(u, u')$ if $\hat{\sigma}(u, u') > 0$, otherwise $W_{\text{ULB}}(u, u') = \theta$, where θ is a very small (close to zero) positive number.

In this method, the function `ComputeUserGraph()` computes the graph G*-ULB. Essentially, G*-ULB enhances G*-LB by introducing the additional edges found in G*-UB, thus addressing the problem of missing edges in G*-LB. Nevertheless, it relies on lower bounds as edge weights, assigning a very small value θ to compensate for those missing (i.e., for those introduced by G*-UB), thus addressing the problem that the upper bounds are typically looser than the lower bounds.

Edge refinement In all three methods presented above, the accuracy of the generated graph can be improved by refining the edge weights, i.e., computing the exact similarity scores. Such a refinement can be performed by executing the PPJ-B algorithm on the corresponding user pair and updating the edge weight with the returned exact similarity score. Given that we want to refine only a subset of the edges (to keep execution time low), the goal is to prioritize edge refinements, so that those performed are the ones that are more likely to contribute in improving the quality of the identified clusters. Next, we present a method for this purpose.

According to the definition of modularity, the assignment of a node u to a cluster c depends on how the weights of the edges between u and the nodes in c compare to the weights of edges between u and the nodes of other candidate clusters. In the three methods using the graphs G*-UB, G*-LB, and G*-ULB, these decisions are based on edge weights that are upper/lower bounds instead of exact similarity scores, thus introducing errors in cluster assignments. The goal of edge refinement is to identify and correct such wrong assignments. Otherwise, if after refining the weights of a node's edges this node still remains in its former cluster, there has been little gain from these refinements.

The intuition for the method presented next is based on the above observation. Specifically, the higher the difference between the modularity gain of the current assignment and that of the second best candidate assignment, the more likely it is that the current assignment will remain valid even after the weights of this node's edges have been refined.

To formalize this criterion, assume a node u, and let $\mathcal{C}_{adj}$ denote the set of clusters to which its adjacent nodes belong, which are also the candidate clusters for assigning u. Recall that, during its first stage, the Louvain method computes for each one of these candidate clusters $c \in \mathcal{C}_{adj}$ the modularity gain $\Delta mod(u, c)$, and assigns u to the one with the maximum gain, assuming that it is positive. Thus, we can assume

Algorithm 6: Procedure `ComputeClusters` for G*-UB, G*-LB and G*-ULB with edge refinement

```
   Input: Approximate graph $G^*$, $\rho$
   Output: Partitioning $P$ assigning users to clusters
1  $P \leftarrow$ InitializeClusters($G^*$)    // singleton clusters
2  $P \leftarrow$ Louvain($G^*$, $P$)    // with stability computations
3  Sort($U$) // in ascending order of stability
4  $refinedEdges = 0$
5  while $refinedEdges < \rho \cdot |E_{G^*}|$ do
6      $u \leftarrow U.getNext()$
7      foreach $e \in u.getUnrefinedEdges()$ do
8          $G^* \leftarrow$ RefineEdgeWeight($G^*$, $e$)
9          $refinedEdges$++
10 ApplyLinearRegression($RefinedEdges$, $RemainingEdges$)
11 $P \leftarrow$Louvain($G^*$, $P$)
12 return $P$
```

that the clusters in $\mathcal{C}_{adj}$ are sorted in descending order of their modularity gain, in which case $\mathcal{C}_{adj}[0]$ corresponds to the candidate that is selected and $\mathcal{C}_{adj}[1]$ corresponds to the second best option. Then, we can define the following value for u:

$$stab(u) = \Delta mod\left(u, \mathcal{C}_{adj}[0]\right) - \Delta mod\left(u, \mathcal{C}_{adj}[1]\right) \quad (14)$$

which is a heuristic measure indicating the stability or confidence of assigning u to the best candidate cluster compared to the second best one.

Once a pass over all nodes has been completed, the nodes are sorted in ascending order of their stability, and edge refinement is performed on the edge weights of each node following this order, up to a specified maximum number of assignments. The latter can be defined as a portion $\rho \in [0, 1]$ of the total number of edges in the graph.

In the methods using the graphs G*-UB, G*-LB, and G*-ULB, when no edge refinements are allowed, the function `ComputeClusters()` simply executes the Louvain method on the constructed graph. Instead, if edge refinements are allowed, the function `ComputeClusters()` applies the procedure outlined in Algorithm 6. This performs a first execution of the Louvain method, during which the stability scores of the nodes are also calculated as explained above (line 2). These are used to prioritize the nodes and refine edge weights accordingly (lines 3 & 8). After refinement, the Louvain method is executed again on the refined graph to compute the user clusters (line 11).

The above process refines only a small subset of the graph edges in order to limit the extra computational cost incurred by these refinements. Still, it is possible to leverage the initial refinements to also adjust the edge weights of all the rest of the edges accordingly. This can be simply accomplished by

Table 1 Datasets used in the experiments

Dataset	Obj.	Users	Distinct Loc.	Kwd/Obj	Obj/Kwd	Obj/User
GeoText	166K	9.5K	32K	1.6	3.5	18
Flickr	1.1M	11.3K	360K	8	26.4	99
Twitter	9.7M	40K	6.8M	2.1	6.3	243

using linear regression [25], where the observations are the lower or upper bounds (or both) and the dependent variable is the actual similarity. Specifically, using the results of the refined edges in the first iteration, a linear regression model is fitted, and it is applied on the rest of the edges to predict the weights based on the known upper/lower bounds (line 10).

6 Experimental evaluation

In this section, we present the experimental evaluation of our methods. First, we describe the datasets used in the experiments. Then, we present the results for each of the two problems addressed, i.e., spatio-textual user matching and clustering.

6.1 Datasets

In our experiments, we have used three real-world datasets containing different types of geotagged posts made by users. These include the following.
GeoText This dataset is a corpus of geotagged microblogs available online[1] [20]. It comprises 377,616 geotagged posts by 9,475 different users within the USA.
Flickr This is derived from the Flickr Creative Commons dataset provided by Yahoo [41]. The whole dataset contains about 99.3 million images, about 49 million of which are geotagged. For our experiments, we have used a subset, containing geotagged photographs with coordinates within a bounding box covering the area of Greater London, UK. This resulted in a dataset containing 11,306 users associated with 1,116,348 geotagged photographs.
Twitter This dataset is a collection of geotagged tweets from the geographical area of Greater London, UK. It is part of the dataset used by [19]. It contains 9,724,579 tweets belonging to 40,000 different users.

For each dataset, we used the NLTK toolkit[2] to extract named entities from the text of the posts. These were then combined with any other keywords (e.g., tags, hashtags) already attached to each post to derive the keyword set associated with each respective spatio-textual object.

The characteristics of the three datasets are summarized in Table 1. For each dataset, the table shows the total number of objects and users it comprises. It also shows the number of distinct locations, as well as the average number of keywords per object, objects per keyword, and objects per user. As can be seen, these datasets differ significantly in size, with GeoText being the smallest and Twitter being the largest one. Moreover, they differ in the characteristics of the objects and the users. For instance, in Flickr, the number of keywords per object as well as the number of objects per keyword is overall much higher. On the other hand, in Twitter, the number of objects per user is considerably higher. Hence, the selection of these datasets for the experiments allows us to test and evaluate our proposed algorithms both in terms of scalability and with respect to different dataset characteristics.

6.2 Results for spatio-textual user matching

First, we evaluate the performance of our algorithms proposed for addressing the spatio-textual user matching problem. The purpose of this evaluation is to compare the efficiency of the proposed algorithms in terms of execution time in different settings. All algorithms were implemented in Java. This set of experiments was 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. In the following, we present the results for each one of the investigated problems.

6.2.1 Comparison for similarity join

We compare the execution time of the three algorithms `S-PPJ-C`, `S-PPJ-B`, and `S-PPJ-F` with respect to the following parameters: (a) the dataset size N in terms of number of users, and (b) the query thresholds for spatial distance (ϵ_{loc}), textual similarity (ϵ_{doc}), and user similarity (ϵ_u).
Effect of dataset size For the first set of experiments, i.e., for testing the scalability of the algorithms, we extract from each dataset a set of smaller subsets, with progressively increasing sizes. Each one of these comprises approximately $N \times 10^3$ users, where $N = [4, 6, 8, 10]$ for GeoText and Flickr, and $N = [10, 20, 30, 40]$ for Twitter.

Moreover, we set the thresholds ϵ_{loc}, ϵ_{doc} and ϵ_u to default values, as shown in Table 2 (note that ϵ_{loc} is measured in decimal degrees, whereas ϵ_{doc} and ϵ_u are ratios). This table also shows, for these default threshold values and each one of

[1] http://www.ark.cs.cmu.edu/GeoText/.

[2] http://www.nltk.org/.

Table 2 Default threshold values and resulting number of matches

	Default thresholds			Number of matches			
	ϵ_{loc}	ϵ_{doc}	ϵ_u	N_1	N_2	N_3	N_4
GeoText	0.001	0.3	0.3	15	23	34	36
Flickr	0.001	0.6	0.6	24	43	72	139
Twitter	0.001	0.4	0.4	5	11	15	23

the dataset subsets, the size of the result set, i.e., the resulting number of matching user pairs.

The reason for not using the same default values across all three datasets is to account for their different characteristics in terms of size, spatial distribution of objects and distribution of keywords. Essentially, we wish to select default values that are both intuitive and produce result sets that are neither too small nor too large. Thus, we set lower thresholds for GeoText in order to avoid empty result sets, whereas higher thresholds are set for Flickr to avoid returning too many matches. The reason that lower thresholds in Flickr tend to produce a much higher number of matches compared to the other two datasets can be attributed to the fact that the spatial and the textual dimensions tend to be more highly correlated in the case of photographs than in the case of tweets or other posts. In other words, usually there is a stronger association between the location of a photograph and its tags compared to that between the location of a tweet and its textual content. Consequently, in the case of the Flickr dataset, users that are more similar in one of these dimensions tend to be also more similar in the other, thus resulting in an overall higher spatio-textual similarity score.

Using the above setting, Fig. 4 shows the performance of the compared algorithms for the different dataset sizes. In all cases, execution time increases as the dataset size increases. However, the results clearly show that `S-PPJ-F` outperforms all other methods by several orders of magnitude, and this result is consistent for all datasets, irrespective of size. This significantly better performance 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.

Regarding the rest of the algorithms, the execution of `S-PPJ-B` is significantly higher than that of `S-PPJ-F`, as already pointed out. This result is expected since `S-PPJ-F` builds on `S-PPJ-B` by leveraging the filter and refinement scheme. Nevertheless, we can observe that `S-PPJ-B` achieves a non-negligible improvement over baseline algorithm `S-PPJ-C`. This allows to assess the benefits of the early termination strategy, as well as the traversal mechanism, that differentiate `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.

Effect of similarity thresholds As already discussed, the selection of the similarity thresholds affects the number of matches among users. In turn, this may also affect the performance of the algorithms. Hence, in the second set of experiments, we evaluate the performance of the proposed algorithms when varying the values of the spatial, textual, and user similarity thresholds ϵ_{loc}, ϵ_{doc}, and ϵ_u. Similar to the scalability experiments, different ranges in threshold values are used across datasets. This set of experiments is performed using the following subsets of the three datasets: GeoText-6K, Flickr-6K, and Twitter-20K. The mean and standard deviation of the result set sizes for the various combinations of threshold values are as follows: 18 (36.9) for GeoText, 326 (633.89) for Flickr, and 14.14 (9.98) for Twitter.

The results are shown in Fig. 5. We observe that the dominant parameter is the spatial threshold ϵ_{loc}. Higher values of

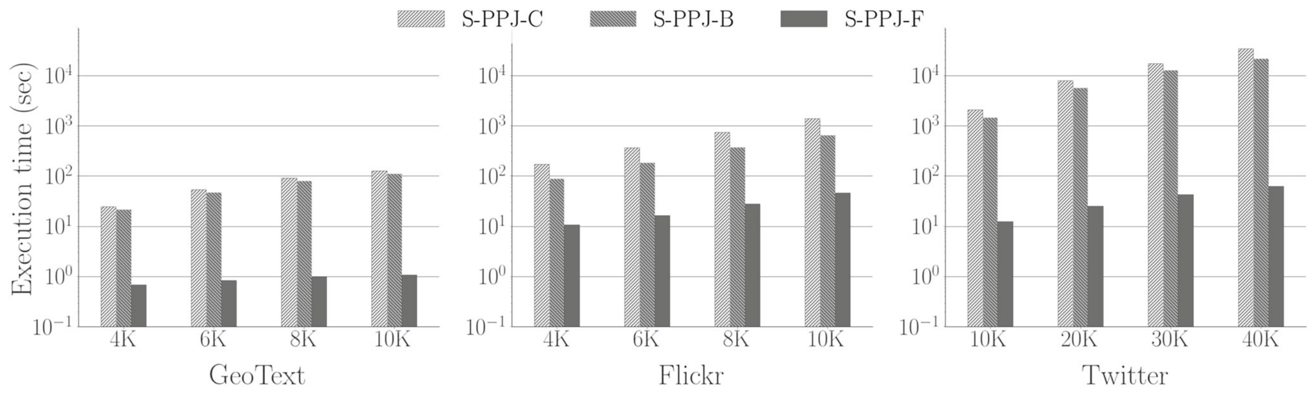


Fig. 4 Comparison of the spatio-textual user join algorithms w.r.t. dataset size

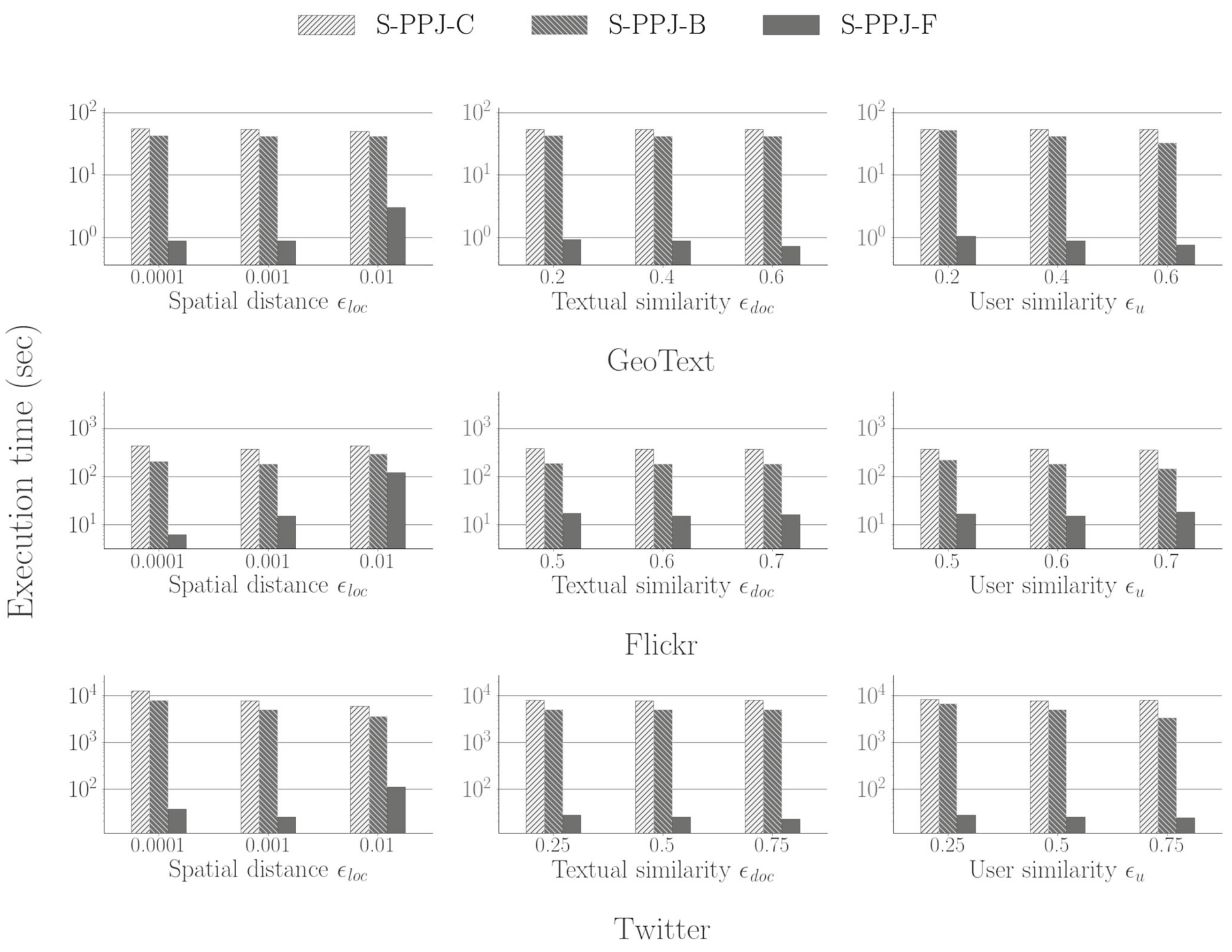


Fig. 5 Comparison of the spatio-textual user join algorithms w.r.t. varying similarity thresholds

ϵ_{loc} result in significantly higher execution times. This is particularly obvious for the Flickr and Twitter datasets, which also contain significantly larger amounts of objects. When the spatial distance threshold reaches the scale of metropolitan level distances, the majority of the objects fall into adjacent partitions. Consequently, the filtering step of `S-PPJ-F` returns a high number of candidates. In these cases, the overhead imposed by the additional indexing maintained by `S-PPJ-F` is apparent.

On the other hand, this does not seem to 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 in this case show that the proposed pruning strategies are highly functional in combination with a grid-based partitioning scheme. Again, `S-PPJ-F` outperforms the other methods in every scenario, and its performance appears to have low sensitivity with respect to the parameter values.

6.2.2 Comparison for top-*k* matches

Next, we compare the performance of `TOPK-S-PPJ-F`, `TOPK-S-PPJ-S`, and `TOPK-S-PPJ-P`, while varying the values of the parameter k. The spatial distance and textual similarity thresholds, ϵ_{loc} and ϵ_{doc}, are set to their default values as shown in Table 2. The results of the experiments are shown in Fig. 6.

Overall, the execution time of all algorithms increases as k increases. Moreover, we can see that `TOPK-S-PPJ-F`, although simpler, is competitive and in fact it even outperforms the others in the Flickr dataset. Compared to that, the higher execution time of `TOPK-S-PPJ-S` indicates that the statistical approach it employs for ordering the users incurs more overhead compared to any potential benefits, thus requiring overall more time than in the case of the simple ordering of the users based exclusively on the size of their object sets, used by the other algorithms. On the other hand, `TOPK-S-PPJ-P` exploits an additional pruning step,

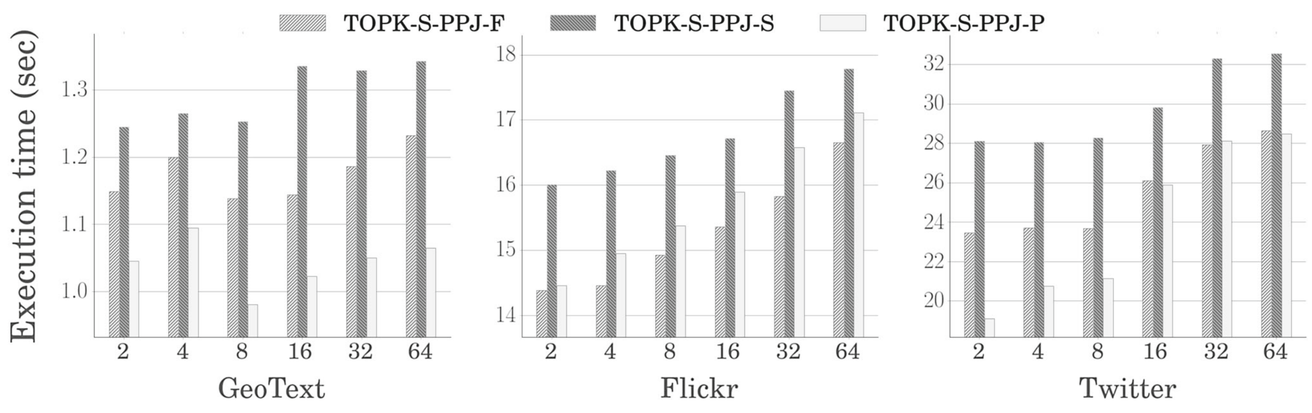


Fig. 6 Comparison of the top-k user pairs algorithms w.r.t. k

which helps to achieve better performance in the cases of GeoText and Twitter. Even in the case of Flickr, where it is outperformed by `TOPK-S-PPJ-F`, it remains competitive. This difference observed in the case of the Flickr dataset is attributed to the fact that it contains objects with higher similarity overall, as already discussed previously. Subsequently, in this case the additional filtering step of `TOPK-S-PPJ-P` does not manage to disqualify large numbers of user pairs.

6.3 Results for spatio-textual user clustering

6.3.1 Illustrative examples

Before delving into the detailed comparison of the proposed methods, we present in Fig. 7 some illustrative examples of the discovered user groups to assess the results. Note that we do not perform a formal evaluation regarding the accuracy of identified user communities. This would require the availability of corresponding ground truth data for comparison and it also depends on other processing steps, in particular data preprocessing and cleaning with NLP and semantic extraction techniques, which are orthogonal to our method and thus beyond the scope of this paper. Moreover, the number of resulting clusters in each case largely depends on the values selected for the spatial distance and textual similarity thresholds. For the default values used in our experiments, the number of identified clusters was 371 for GeoText, 74 for Flickr, and 319 for Twitter; however, only few of them (about 10–15 in each case) contain more than 10 users. Nevertheless, even without much emphasis on data preprocessing and parameter tuning, the presented examples below show that the proposed method has the potential to discover meaningful user groups in the input data.

Figure 7 shows four user groups extracted from the Flickr dataset which contains about 1M geotagged photographs from about 10K users in the area of Greater London. The first group (C_1—top left) appears to correspond to football fans, having posts with keywords such as *football*, *Arsenal*, and *Premiership*, in locations around Wembley Stadium (northwest), Emirates Stadium (north), Boleyn Ground (northeast), and Crystal Palace Park (south). The second group (C_2—top right) appears to refer to users attending tennis events; the posts contain keywords such as *tennis*, *Wimbledon*, and *ATP* and are concentrated around Queen's Club (northwest) and Wimbledon Park (southwest). The third group (C_3—bottom left) appears to comprise tourists that visit various attractions in the center of London, with posts containing keywords such as *St Paul's*, *Westminster*, and *Hungerford Bridge* and being located around these well-known landmarks as well as other ones such as Big Ben and Trafalgar Square. Finally, the fourth group (C_4—bottom right) involves users that appear to have some cultural interests, containing posts with keywords such as *architecture*, *museum*, and *statue* in locations including the British Museum, the British Library, the London Transport Museum, and the Greenwich Park.

6.3.2 Method comparison

Having examined some illustrative cases of the resulting user clusters, we proceed with the comparison of our proposed algorithms for this task. The experiments presented next were executed on an Intel Xeon E5-2420 v2 CPU with 2.20 GHz processor and 64GB RAM running Ubuntu. The spatial distance and textual similarity thresholds, ϵ_{loc} and ϵ_{doc}, are set to the same default values used previously (see Table 2).

For brevity, we distinguish between the compared methods by indicating the type of graph they use. For example, we refer to the method generating the approximate graph `G`*`-TOPK` and then applying the Louvain method on it, simply as `G`*`-TOPK`. Moreover, for those methods where edge weight refinement is applicable, we indicate the type of refinement used (if any). For example, `G`*`-LB (R1)` indicates the method that generates the graph `G`*`-LB` and then, during cluster computation, utilizes edge weight refinement according to the method `R1`, as outlined in Algorithm 6.

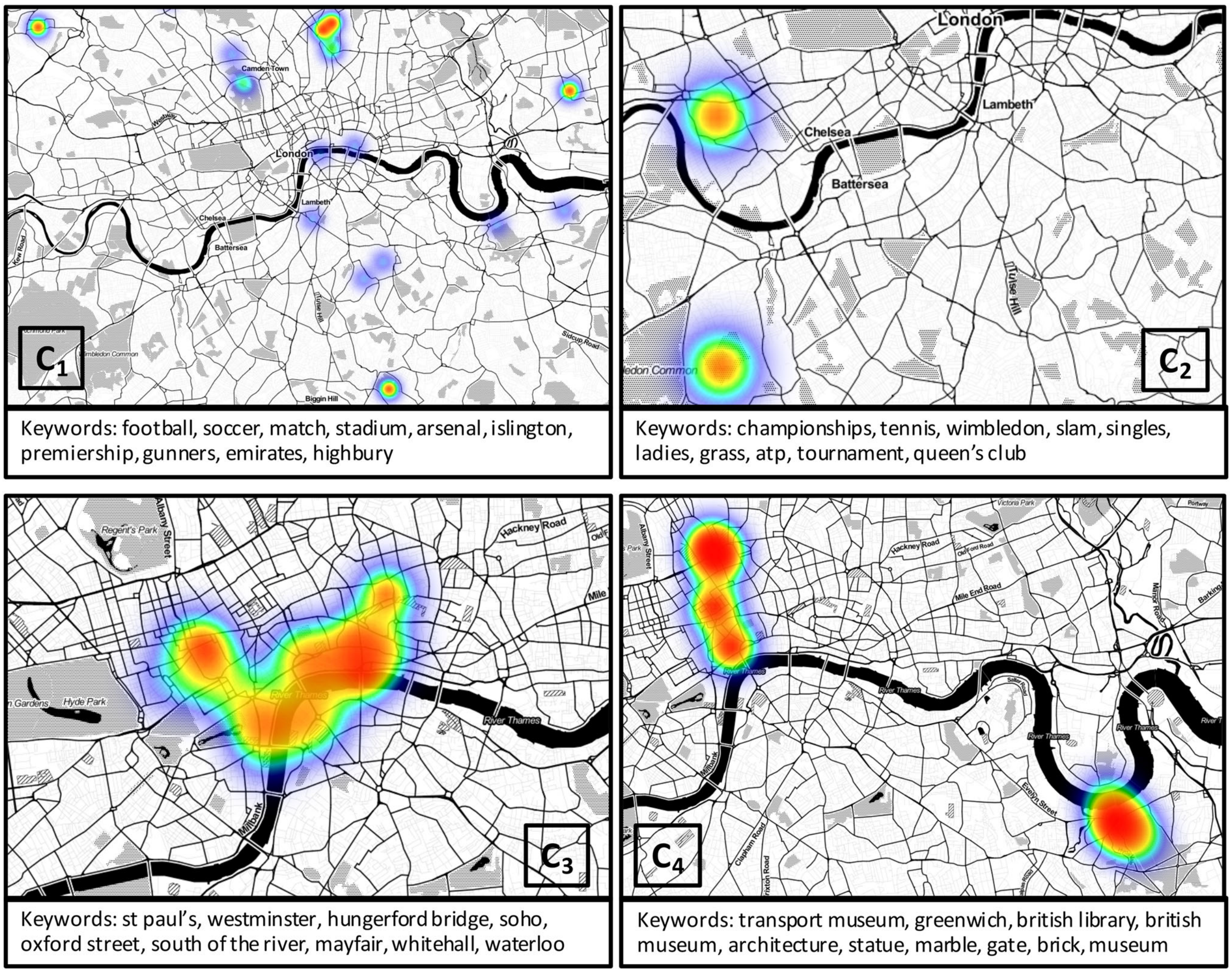

C1
Keywords: football, soccer, match, stadium, arsenal, islington, premiership, gunners, emirates, highbury
C2
Keywords: championships, tennis, wimbledon, slam, singles, ladies, grass, atp, tournament, queen's club
C3
Keywords: st paul's, westminster, hungerford bridge, soho, oxford street, south of the river, mayfair, whitehall, waterloo
C4
Keywords: transport museum, greenwich, british library, british museum, architecture, statue, marble, gate, brick, museum

Fig. 7 Illustrative example showing the spatial footprint and top keywords for four identified user groups

In the evaluation, we compare the performance of the methods relying on the approximate graphs G^* as opposed to that of the exact method, which fully computes the spatio-textual user similarity graph G and then executes the Louvain method over it. We denote as T_{ex}, M_{ex}, and C_{ex}, respectively, the execution time of the exact method, the modularity score of the clusters it produces, and the set of these clusters. For each approximate method, we measure, on the one hand, its efficiency compared to T_{ex}, and, on the other hand, its quality compared to M_{ex} and C_{ex}. In particular, for the quality comparison, we use two criteria. The first is based on the modularity scores of the resulting partitions. For each approximate method, we apply the resulting partitions on the exact graph G and compare the obtained modularity score to M_{ex}. This shows to what extent each method can successfully identify a community structure in the original graph. It is possible for two sets of partitions to have similar modularity scores while being quite different (i.e., there may be more than one relatively good ways to group together users into communities). Thus, we use in addition a second measure that indicates how similar the partitions are. In particular, we compute the normalized mutual information (NMI) [40] between the set of clusters obtained by each method and the set of clusters computed on the exact similarity graph.

Finally, we study the trade-off provided by each method between execution time and quality of results.

6.3.3 Baseline

In addition to comparing our proposed methods, we also examine the performance of a baseline method, where the idea is to quickly approximate user similarities based on dimensionality reduction. First, we map all objects to a set of discrete locations. Specifically, we use k-means to cluster objects according to their coordinates, and then, we map each object to the centroid of the cluster it is assigned to. Next,

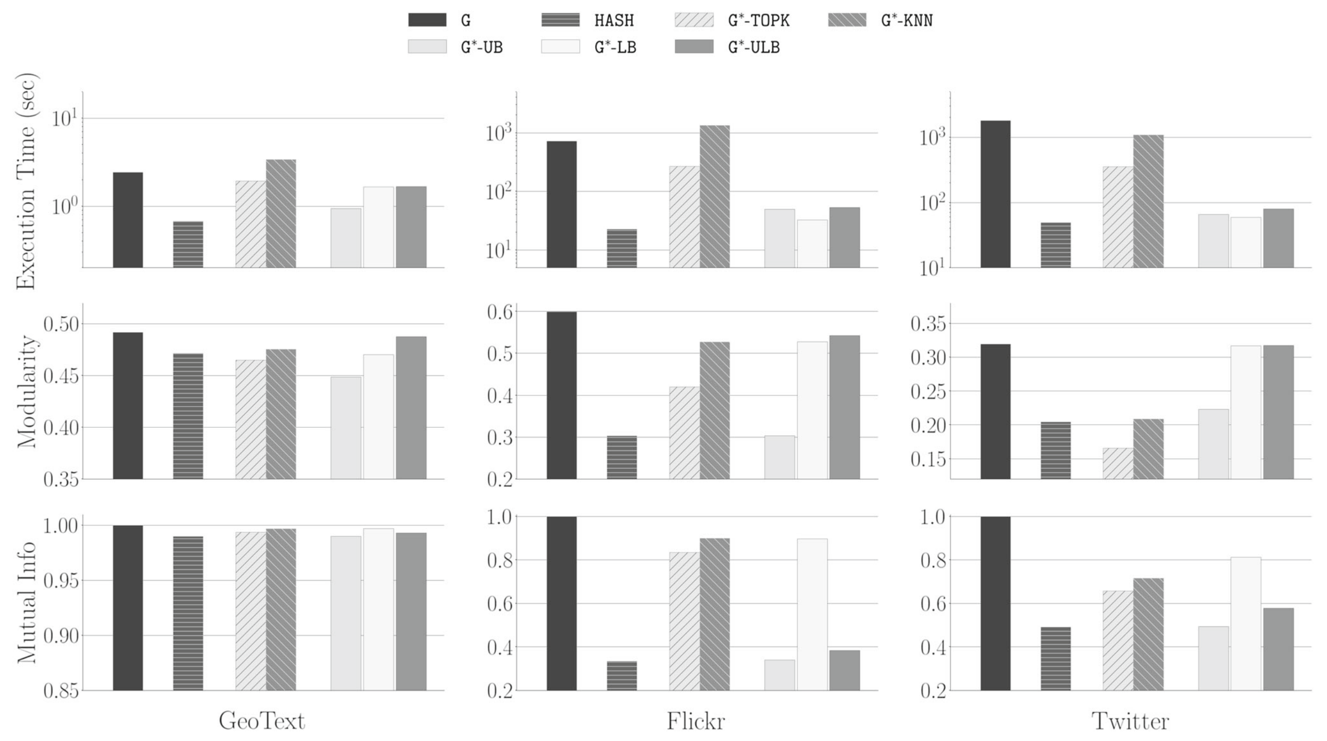


Fig. 8 Comparison of the proposed methods for spatio-textual user clustering

we transform the keywords of each post to spatio-textual "words", by prefixing them with the identifier of the respective cluster centroid. Finally, we apply Twitter-LDA [50], an adaptation of LDA for microblogs, to extract a set of τ spatio-textual "topics" and represent each user with a vector in this τ-dimensional space.

Using these user vectors, we have investigated two methods for computing user groups. The one is a straightforward approach where locality-sensitive hashing (LSH) [26] is applied to map similar users to "buckets". This approach allows to group together users very quickly (in the order of milliseconds); however, in our experiments, the resulting partitions had very low quality. Measuring the modularity of the obtained partitions on the exact user similarity graph, the resulting values were very low (close to zero or even negative), which indicates that the method fails to identify communities in the graph. Therefore, we also investigate a second alternative that operates in a similar manner as the rest of our proposed algorithms (i.e., it constructs a user similarity graph and executes the Louvain method to identify communities), with the difference that in this case the similarity between a pair of users is computed by simply calculating the cosine similarity of their respective vectors, as opposed to performing a detailed comparison of their spatio-textual objects. In the experiments presented in the next section, we refer to this baseline approach as HASH. Note that, for this baseline, the measured execution time does not include the time needed to cluster locations using k-means and to extract spatio-textual topics using Twitter-LDA. These are considered as preprocessing steps that are executed offline.

6.3.4 Overall comparison of methods

The first set of experiments provides an overview regarding how the different proposed methods compare to each other in terms of both execution time and quality of discovered clusters. In this experiment, for G*-TOPK, we set the parameter k to 4000 for GeoText, 10,000 for Flickr, and 40,000 for Twitter, to account for their differences in size and complexity. For G*-KNN, we set the parameter k to 5. More detailed results with varying values of these parameters are presented in Sect. 6.3.5. Moreover, for the baseline method, we set the number of location clusters and topics to $k = 100$ and $\tau = 100$, respectively. The results are shown in Fig. 8.

All methods, except G*-KNN, require only a fraction of the execution time T_{ex} of the exact method. This is particularly evident in the case of the largest dataset (Twitter). In that case, the execution time required by G*-TOPK and G*-KNN is around 20 and 60% of T_{ex}, respectively. This reduction is even higher for the methods based on G*-UB, G*-LB, and G*-ULB, which manage to reduce the execution time to less than 5% of T_{ex}. We observe similar results for the Flickr dataset for almost all algorithms. G*-TOPK requires 37% of T_{ex}, whereas G*-UB, G*-LB, and G*-ULB achieve execution times less than 8% of T_{ex}. However, G*-KNN exhibits poor performance, even exceeding T_{ex} in the case of GeoText and

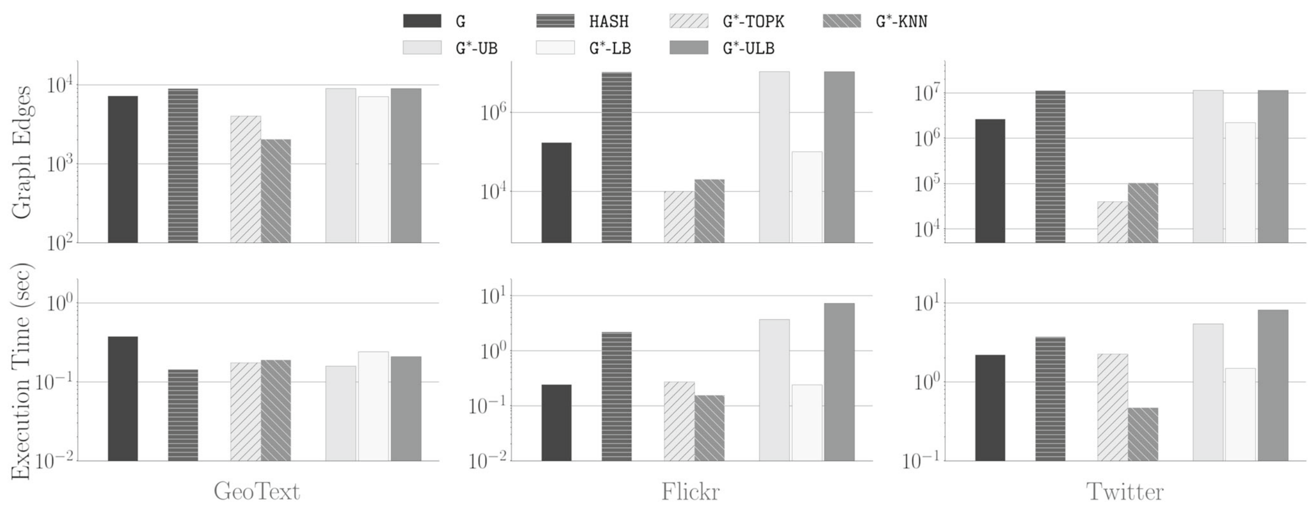


Fig. 9 Comparison of sizes of the generated graphs, and of the time spent for computing clusters in each one

Flickr. This can be attributed to the fact that, since the execution of the kNN query has to be done for each individual user, eventually this results in many repeated computations accumulating an overhead that negates any benefits of the pruning performed within each individual iteration. Overall, focusing on the two most demanding datasets, Flickr and Twitter, we can conclude that the most efficient method is G*-LB. Still, the baseline method executes faster than any other algorithm. This is expected, since in this case user similarities are measured by simply computing the cosine similarities between the respective vectors that have been constructed offline. However, as shown next, this method identifies clusters of lower quality.

In terms of quality, G*-LB and G*-ULB achieve the best results overall. The modularity of the clusters they produce is comparable to M_{ex} in GeoText and Twitter and also quite close to M_{ex} in Flickr. Interestingly, G*-KNN also produces clusters of equally good quality in Flickr, but, as discussed above, at a much higher cost. When examining the respective results for the normalized mutual information, a difference can be observed in the behavior of the G*-ULB algorithm. The scores in this case are quite low, which indicates that G*-ULB partitions the user graph in a different manner than G; however, as seen above, this partition still seems to be of high quality, as indicated by the high modularity score. Instead, G*-LB achieves a high score both in modularity and in normalized mutual information, indicating that the produced partitioning more closely matches that of G. Finally, the baseline method exhibits poor results, in terms of both modularity and normalized mutual information, except from the GeoText dataset. Notice that these scores remained low even when testing different values of the parameters k and τ.

Overall, G*-LB and G*-ULB achieve the best combined performance in efficiency and effectiveness. G*-ULB achieves marginally higher modularity scores, although it appears to identify different user groups, whereas G*-LB has marginally lower execution time, while also finding user groups that are more similar to the one discovered on the exact user graph. This outcome is attributed to the effectiveness of G*-LB in more closely approximating the exact user similarity graph. G*-ULB requires slightly more time to compute, but it also introduces additional edges which are missed in G*-LB. It appears that some of these edges, even with their very small weight θ, contribute to deriving better clusters. On the other hand, although G*-UB has a comparable performance to G*-LB and G*-ULB in terms of execution time, it fails to produce clusters of high quality. This is attributed to the fact that the upper bound is a quite loose approximation of the actual similarity score. The outcome is similar for G*-TOPK. Despite the reduction in execution time, the resulting clusters turn out to have poor quality. This shows that focusing only on the top-k connections is not sufficient.

To delve into some more detail regarding the graphs generated by each method, we measure the size of each one (in terms of the number of edges) as well as the time needed for computing the clusters over it. The results are shown in Fig. 9. In the case of G*-TOPK and G*-KNN, the number of produced edges is explicitly controlled by the respective parameter k; thus, we focus more on the other methods. G*-LB generates a graph that has fewer edges than G, whereas the number of edges in the graph generated by G*-UB (and, consequently, also by G*-ULB) is considerably higher. This indicates the ratio of false positive and false negative edges in the respective methods. Subsequently, it results in more time required for graph partitioning in G*-UB and G*-ULB compared to G*-LB, in particular for Flickr and Twitter. The behavior of the baseline method is also close to that of G*-UB and G*-ULB. Still, this difference has negligible impact on the overall execution time, since the time spent on graph partitioning remains a small fraction of it. The latter is shown in

Table 3 Graph partitioning time to total execution time

	G (%)	G*-TOPK (%)	G*-KNN (%)	G*-UB (%)	G*-LB (%)	G*-ULB (%)
GeoText	15.6	9.1	5.55	17	15	12
Flickr	0.03	0.1	0.01	7.5	1	14
Twitter	0.12	0.6	0.04	8.3	3	10

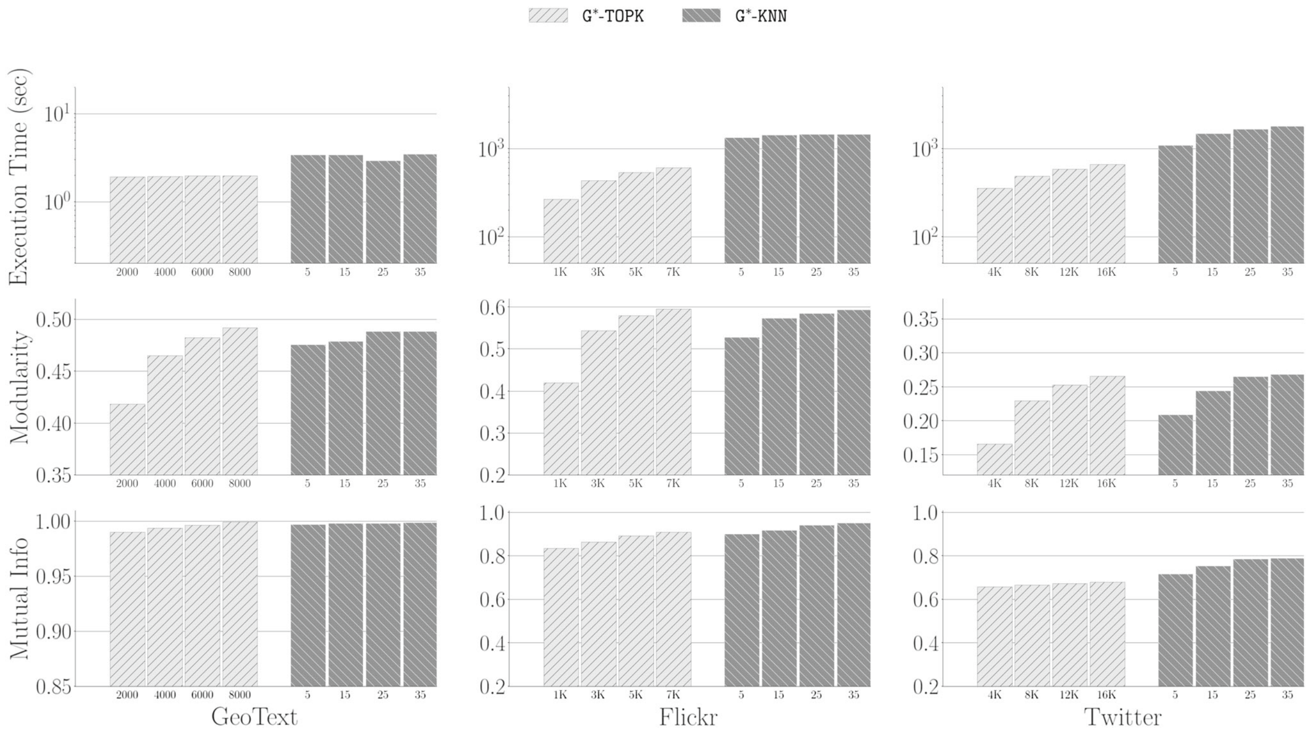


Fig. 10 Performance of G*-TOPK and G*-KNN while varying k

more detail in Table 3. Even in the cases of the larger graphs G*-UB and G*-ULB, the time spent on graph partitioning does not exceed 17% of the total execution time. This observation advocates in favor of strategies that focus on reducing the time required by the first phase of the process, namely the graph generation, which involves the costly set similarity join operations.

6.3.5 Effect of parameter *k* (G*-TOPK and G*-KNN)

In the G*-TOPK and G*-KNN algorithms, the respective parameter k allows for a means to control the balance between execution time and quality of results. In the next experiment, we investigate this behavior. Specifically, for G*-TOPK, we vary k from 2000 to 8000 for GeoText; from 10,000 to 70,000 for Flickr; and from 40,000 to 160,000 for Twitter. For G*-KNN, we vary k from 5 to 35. The results are shown in Fig. 10.

In the GeoText dataset, the resulting differences in execution time are not very noticeable; however, the execution time in this dataset is relatively small. In contrast, when examining the Flickr and Twitter datasets, the execution time clearly increases as k increases. Interestingly, in Twitter, G*-KNN achieves lower execution times than T_{ex} for smaller values of k, but in all other cases it exhibits poor performance, as already observed previously. Examining the modularity of the produced clusters, we can also observe a clear increase in its score as the value of k increases. This shows that as the threshold on the number of edges to be computed is relaxed, the quality of the identified clusters is improved, thus validating the trade-off between efficiency and accuracy. The same holds for the normalized mutual information. Nevertheless, both methods manage to achieve comparable modularity to M_{ex} only when k reaches higher values, in which case the execution time has also reached levels comparable to T_{ex}.

6.3.6 Effect of edge refinement (G*-UB, G*-LB, G*-ULB)

Our final set of experiments investigates the performance of the methods based on G*-UB, G*-LB, and G*-ULB, when edge weight refinement is enabled. In this case, an initial graph partitioning is first computed. Then, the algorithm selects

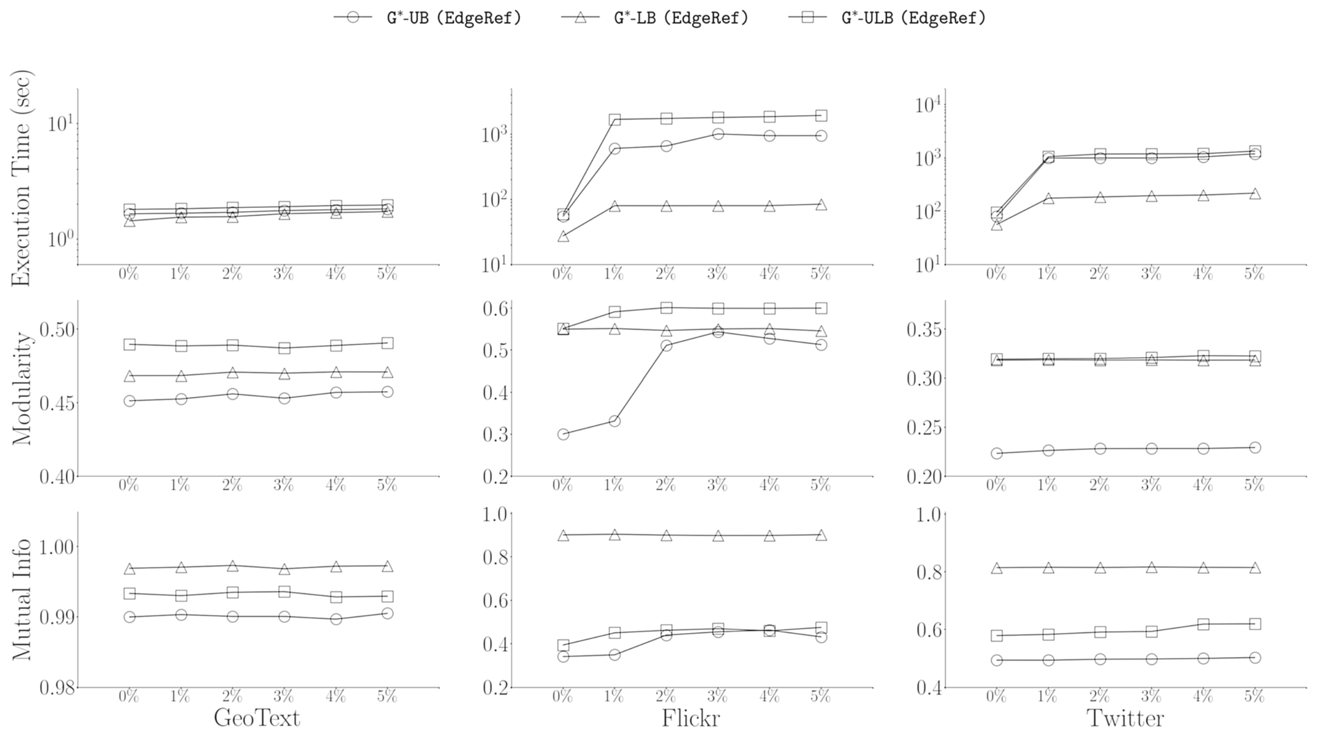


Fig. 11 Performance of G*-UB, G*-LB, and G*-ULB with edge weight refinement

and refines a (small) subset of the graph edges, prioritized according to the stability heuristic. Moreover, it leverages these refined results to accordingly adjust the weights of the rest of the edges, through the application of a linear regression model. Afterward, it executes again the Louvain method to compute a new partitioning on the refined graph. These steps of edge refinement and graph partitioning are applied iteratively, up to a maximum number of iterations or until the process converges. In this experiment, we evaluate the performance of each method while varying the percentage of edges that are refined.

The results are shown in Fig. 11. Overall, we can observe that in the case of G*-LB, applying edge weight refinement gradually increases the execution time, without offering a corresponding increase in the quality of the produced clusters. Nevertheless, as we have already seen, the modularity scores and the normalized mutual information scores achieved by G*-LB even without any edge refinement are already comparable to those achieved when operating on the exact user similarity graph. On the other hand, when edge weight refinement is applied on G*-UB and G*-ULB, the increase in execution time is accompanied by some increase in cluster quality. This is especially true when considering the impact of applying edge refinement on G*-UB, in particular on the Flickr dataset. In that case, modularity is drastically improved, reaching the levels of G*-LB, although at the cost of a higher execution time. The benefits of edge refinement can also be observed when applied to G*-ULB, in which case it increases its modularity scores to even higher values than G*-LB, while still retaining reduced execution times, around 70–90% of T_{ex}. The impact of edge refinement is less noticeable on the Twitter dataset, where some improvements are still observed when refining 4–5% of the edges.

7 Conclusions

In this paper, we have addressed the problem of spatio-textual user matching and clustering, where each user is associated with a set of spatio-textual objects. For user matching, we have presented several algorithms that rely on spatio-textual indexing and pruning techniques to process an STPSJoin query efficiently. We have also extended and adapted these algorithms to the top-k variant of the problem. User clustering is then based on generating and partitioning a spatio-textual user similarity graph. To speed up this process, our proposed techniques rely on building a partial and approximate similarity graph instead. All algorithms are evaluated experimentally on three real-world datasets comprising geolocated user posts and photographs. For the matching problem, the results indicate that S-PPJ-F performs best, while TOPK-S-PPJ-P achieves the best performance for the top-k variant in the majority of the experiments. For the clustering problem, our methods relying on the G*-LB and G*-ULB approximate graphs are the most effective, drastically reducing the execution time while still producing clusters of high quality.

Our future work focuses on two main directions. The first is to focus more on the quality of the produced user clusters, investigating different community detection algorithms. The second is to further improve the efficiency and scalability of our methods by proposing parallel and distributed algorithms for these problems.

References

1. Adelfio, M.D., Nutanong, S., Samet, H.: Searching web documents as location sets. In: SIGSPATIAL, pp. 489–492 (2011a)
2. Adelfio, M.D., Nutanong, S., Samet, H.: Similarity search on a large collection of point sets. In: SIGSPATIAL, pp. 132–141 (2011b)
3. Aynaud, T., Blondel, V.D., Guillaume, J.-L., Lambiotte, R.: Multi-level local optimization of modularity. In: Bichot, C.-E., Siarry, P. (eds.) Graph Partitioning. Wiley, Hoboken, NJ (2013). https://doi.org/10.1002/9781118601181.ch13
4. Ballesteros, J., Cary, A., Rishe, N.: SpSJoin: parallel spatial similarity joins. In: SIGSPATIAL, pp. 481–484 (2011)
5. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: WWW, pp. 131–140 (2007)
6. Bichot, C.E., Siarry, P.: Graph Partitioning. Wiley, New York (2013)
7. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), P10008 (2008)
8. Bouros, P., Ge, S., Mamoulis, N.: Spatio-textual similarity joins. PVLDB **6**(1), 1–12 (2012)
9. Brinkhoff, T., Kriegel, H., Seeger, B.: Efficient processing of spatial joins using r-trees. In: SIGMOD, pp. 237–246 (1993)
10. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. CoRR abs/1311.3144 (2013)
11. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: ICDE, p. 5 (2006)
12. Chen, L., Cong, G., Jensen, C.S., Wu, D.: Spatial keyword query processing: an experimental evaluation. PVLDB **6**(3), 217–228 (2013)
13. Chen, Y., Suel, T., Markowetz, A.: Efficient query processing in geographic web search engines. In: SIGMOD, pp. 277–288 (2006)
14. Chen, Y., Xu, J., Xu, M.: Finding community structure in spatially constrained complex networks. Int. J. Geogr. Inf. Sci. **29**(6), 889–911 (2015)
15. Christoforaki, M., He, J., Dimopoulos, C., Markowetz, A., Suel, T.: Text versus space: efficient geo-search query processing. In: CIKM, pp. 423–432 (2011)
16. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. Phys. Rev. E **70**(6), 066,111 (2004)
17. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. PVLDB **2**(1), 337–348 (2009)
18. Efstathiades, C., Belesiotis, A., Skoutas, D., Pfoser, D.: Similarity search on spatio-textual point sets. In: EDBT, pp. 329–340 (2016)
19. Efstathiades, H., Antoniades, D., Pallis, G., Dikaiakos, M.D.: Identification of key locations based on online social network activity. In: ASONAM, pp. 218–225 (2015)
20. Eisenstein, J., O'Connor, B., Smith, N.A., Xing, E.P.: A latent variable model for geographic lexical variation. In: EMNLP, pp. 1277–1287 (2010)
21. Expert, P., Evans, T.S., Blondel, V.D., Lambiotte, R.: Uncovering space-independent communities in spatial networks. Proc. Natl. Acad. Sci. **108**(19), 7663–7668 (2011)
22. Fan, J., Li, G., Zhou, L., Chen, S., Hu, J.: SEAL: spatio-textual similarity search. PVLDB **5**(9), 824–835 (2012)
23. Fang, Y., Cheng, R., Li, X., Luo, S., Hu, J.: Effective community search over large spatial graphs. PVLDB **10**(6), 709–720 (2017)
24. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE, pp. 656–665 (2008)
25. Freedman, D.A.: Statistical Models: Theory and Practice. Cambridge University Press, Cambridge (2009)
26. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB, pp. 518–529 (1999)
27. Jacox, E.H., Samet, H.: Spatial join techniques. TODS **32**(1), 7 (2007)
28. Jiang, Y., Li, G., Feng, J., Li, W.: String similarity joins: an experimental evaluation. PVLDB **7**(8), 625–636 (2014)
29. Liu, S., Li, G., Feng, J.: Star-join: spatio-textual similarity join. In: CIKM, pp. 2194–2198 (2012)
30. Liu, S., Li, G., Feng, J.: A prefix-filter based method for spatio-textual similarity join. TKDE **26**(10), 2354–2367 (2014)
31. Newman, M.E.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
32. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**(2), 026,113 (2004)
33. Onnela, J.P., Arbesman, S., González, M.C., Barabási, A.L., Christakis, N.A.: Geographic constraints on social network groups. PLoS ONE **6**(4), e16,939 (2011)
34. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP operations in spatial data warehouses. In: SSTD, pp. 443–459 (2001)
35. Pons, P., Latapy, M.: Computing communities in large networks using random walks. J. Graph Algorithms Appl. **10**(2), 191–218 (2006)
36. Rao, J., Lin, J.J., Samet, H.: Partitioning strategies for spatio-textual similarity join. In: SIGSPATIAL, pp. 40–49 (2014)
37. Rocha-Junior, J.B., Gkorgkas, O., Jonassen, S., Nørvåg, K.: Efficient processing of top-k spatial keyword queries. In: SSTD, pp. 205–222 (2011)
38. Sarawagi, S., Kirpal, A.: Efficient set joins on similarity predicates. In: SIGMOD, pp. 743–754 (2004)
39. Schaeffer, S.E.: Survey: graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007)
40. Strehl, A., Ghosh, J.: Cluster ensembles—a knowledge reuse framework for combining multiple partitions. J. Mach. Learn. Res. JMLR **3**, 583–617 (2002)
41. Thomee, B., Shamma, D.A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., Li, L.J.: The new data and new challenges in multimedia research. arXiv preprint arXiv:1503.01817 (2015)
42. Vaid, S., Jones, C.B., Joho, H., Sanderson, M.: Spatio-textual indexing for geographical search on the web. In: SSTD, pp. 218–235 (2005)
43. Wakita, K., Tsurumi, T.: Finding community structure in mega-scale social networks. CoRR abs/cs/0702048 (2007)
44. Wang, J., Li, G., Feng, J.: Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In: SIGMOD, pp. 85–96 (2012)
45. Wu, D., Cong, G., Jensen, C.S.: A framework for efficient spatial web object retrieval. VLDB J. **21**(6), 797–822 (2012)
46. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient similarity joins for near-duplicate detection. TODS **36**(3), 15 (2011)
47. Zhang, D., Tan, K., Tung, A.K.H.: Scalable top-k spatial keyword search. In: EDBT, pp. 359–370 (2013)
48. Zhang, D., Chan, C., Tan, K.: Processing spatial keyword query as a top-k aggregation query. In: SIGIR, pp. 355–364 (2014a)
49. Zhang, Y., Ma, Y., Meng, X.: Efficient spatio-textual similarity join using mapreduce. In: WI-IAT, pp. 52–59 (2014b)
50. Zhao, W.X., Jiang, J., He, J., Song, Y., Achananuparp, P., Lim, E., Li, X.: Topical keyphrase extraction from twitter. In: ACL, pp. 379–388 (2011)
51. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.: Hybrid index structures for location-based web search. In: CIKM, pp. 155–162 (2005)