A Novel Indexing Method for Spatial-Keyword Range Queries

Panagiotis Tampakis
University of Piraeus
Piraeus, Greece
ptampak@unipi.gr

Dimitris Spyrellis
University of Piraeus
Piraeus, Greece
dim.spyrelis@gmail.com

Christos Doulkeridis
University of Piraeus
Piraeus, Greece
cdoulk@unipi.gr

Nikos Pelekis
University of Piraeus
Piraeus, Greece
npelekis@unipi.gr

Christos Kalyvas
University of the Aegean
Karlovasi, Greece
chkalyvas@aegean.gr

Akrivi Vlachou
University of the Aegean
Karlovasi, Greece
avlachou@aegean.gr

ABSTRACT

Spatial-keyword queries are important for a wide range of applications that retrieve data based on a combination of keyword search and spatial constraints. However, efficient processing of spatial-keyword queries is not a trivial task because the combination of textual and spatial data results in a high-dimensional representation that is challenging to index effectively. To address this problem, in this paper, we propose a novel indexing scheme for efficient support of spatial-keyword range queries. At the heart of our approach lies a carefully-designed mapping of spatio-textual data to a two-dimensional (2D) space that produces compact partitions of spatio-textual data. In turn, the mapped 2D data can be indexed effectively by traditional spatial data structures, such as an R-tree. We propose bounds, theoretically proven for correctness, that lead to the design of a filter-and-refine algorithm that prunes the search space effectively. In this way, our approach for spatial-keyword range queries is readily applicable to any database system that provides spatial support. In our experimental evaluation, we demonstrate how our algorithm can be implemented over PostgreSQL and exploit its underlying spatial index provided by PostGIS, in order to process spatial-keyword range queries efficiently. Moreover, we show that our solution outperforms different competitor approaches.

CCS CONCEPTS

• **Information systems** → **Data structures**; *Geographic information systems.*

KEYWORDS

Spatial keyword search, Data transformation, Spatial data

ACM Reference Format:
Panagiotis Tampakis, Dimitris Spyrellis, Christos Doulkeridis, Nikos Pelekis, Christos Kalyvas, and Akrivi Vlachou. 2021. A Novel Indexing Method for Spatial-Keyword Range Queries. In *17th International Symposium on Spatial and Temporal Databases (SSTD '21), August 23–25, 2021, virtual, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3469830.3470897

SSTD '21, August 23–25, 2021, virtual, USA

ACM ISBN 978-1-4503-8425-4/21/08…$15.00
https://doi.org/10.1145/3469830.3470897

1 INTRODUCTION

During the last decade, the combination of GPS-enabled mobile devices, mobile internet and social networking has led to the generation of huge spatio-textual data sets, where data objects contain both location information as well as textual descriptions. Twitter and Flickr are prominent examples of social networking applications that collect geotagged content daily, in the form of short messages and photos respectively. On the other hand, the amount of geotagged content on the Web has also increased dramatically. For example, points-of-interest (e.g., hotels, restaurants, etc.) are geotagged and some studies [8] have reported that approximately 20% of all Web queries also have location constraints, i.e., also refer to the location of a geotagged web page. As a result, scalable management and querying of spatio-textual data has attracted a lot of attention recently [3, 4, 6, 17].

In order to support efficient retrieval of spatio-textual data, spatial-keyword queries are used and several efficient query processing algorithms have been proposed. In this work, we focus on *spatial-keyword range queries*, where given a query q that consists of a location $(q.x, q.y)$ and a set of keywords Q, the objective is to *retrieve all spatio-textual objects within distance r from q and having keyword set similarity above a user-specified threshold τ*. This query type offers more flexibility than boolean spatial-keyword range queries [5, 7, 13, 15] that impose exact matching on the keywords that describe each object. Nevertheless, efficient processing of spatial-keyword queries requires the use of specialized access methods [8, 9, 14, 16, 18, 20, 22] that combine spatial and text indexing techniques in a joint index, a challenging topic due to the high dimensionality of spatio-textual representations. Unfortunately, these index structures typically have high memory or disk requirements due to the integration of spatial with textual information, and moreover they are not supported in existing database management systems.

Motivated by this limitation, in this work, we propose a novel indexing method for spatio-textual data that supports efficient processing of spatial-keyword range queries. In contrast to existing approaches that devise new index structures, we propose a carefully-designed mapping of spatio-textual data to a two-dimensional (2D) space, where one dimension is used to represent spatial distance and the other textual similarity. Our work is inspired by the iDistance technique [11, 21] from spatial databases that indexes the distance rather than the actual location of objects. One of the main technical challenges addressed in this paper is how to map the

keyword descriptions to 1D values in a way that supports efficient retrieval. Intuitively, the proposed mapping generates data partitions in the 2D space that contain objects with small distances and common keywords, thus preserving data locality. As a result, any traditional spatial index, such as an R-tree, can be used to index the transformed data, thus alleviating the need for specialized index structures. Furthermore, our approach is directly applicable to existing database management systems that provide built-in 2D indexes, such as PostGIS, MySQL, Oracle, etc.

Capitalizing on the proposed mapping, we propose a novel method for processing spatial-keyword range queries. First, we prove the existence of lower and upper bounds for search, which can be exploited in order to derive an efficient processing algorithm that prunes the search space. Then, we present a filter-and-refinement algorithm, called *ST2D*, for spatial-keyword range queries that issues a set of window queries in the transformed 2D space and is provably correct. Moreover, we present an improved version, called *PPR_ST2D*, which uses proximity-aware partition reordering in order to process a single window query and return the correct result set. We implement our algorithms on top of PostgreSQL to show the feasibility of our approach, and we show the gain in terms of performance. Also, in our experiments, we compare against other mappings for spatio-textual data, such as [15] that generates 1D values, and show the benefits of our approach.

In brief, we make the following contributions in this paper:

- We propose a novel method that transforms spatio-textual object to points in a 2D space, which can be effectively indexed using traditional spatial access methods.
- We provide appropriate search bounds for spatial-keyword range queries in the transformed space, in order to prune the search space and avoid processing unnecessary data objects, and prove the correctness.
- We develop an algorithm (*ST2D*) for query processing of spatial-keyword range queries in the transformed space using multiple window queries, as well as an extension (*PPR_ST2D*) that improves data locality in the spatial dimension and requires a single window query.
- We demonstrate the efficiency of our algorithms, implemented in PostgreSQL, in comparison with suitable competitors using two real-life data sets.

The rest of this paper is structured as follows: Section 2 provides an overview of our approach. Then, in Section 3, we describe the mapping scheme which enables representing spatio-textual data in a 2D space. Section 4 presents the query processing algorithm *ST2D*, together with appropriate bounds for the search space that guarantee correctness. Section 5 presents an extension called *PPR_ST2D* that improves data locality in the spatial dimension. Section 6 reports our experimental evaluation, while Section 7 describes the related work. Finally, we conclude the paper and sketch future research directions in Section 8.

2 OVERVIEW

Consider a data set D of spatio-textual data objects, where each object p is associated with a spatial location $(p.x, p.y)$ as well as a set of keywords (tags) denoted with P. We use capital letters (P, Q) to denote the keyword sets associated with an object (p, q).

Symbol	Description
D	Data set of spatio-textual objects
$p \in D$	Spatio-textual object, e.g., $p = \{p.x, p.y, P\}$
q	Query object $q = \{q.x, q.y, Q\}$
P, Q	Sets of keywords for object p, q (resp.)
r	Distance threshold
τ	Textual similarity threshold
$C_i(K_i, r_i)$	Spatial cluster with centroid K_i and radius r_i
$V = \{t_1, t_2, \ldots, t_{\|V\|}\}$	Vocabulary of keywords in D
V_i, V_j	Disjoint subsets of V
c, c'	Constants used by iDistance
$\|C\|$	Number of spatial partitions
k	Number of text partitions

Table 1: Table of symbols.

Given a query q that consists of a location $(q.x, q.y)$ and a set of keywords Q, the *spatial-keyword range query* retrieves all objects within distance r and having keyword set similarity above τ, as defined in the following.

Definition 2.1. (Spatial-keyword range query): Given a query point q, a spatial range r, a set of query keywords Q, and a textual similarity threshold τ, the spatial-keyword range query retrieves all spatio-textual objects $p \in D$, such that: $\text{dist}(p, q) \leq r$ and $\text{sim}(P, Q) \geq \tau$.

In this work, we use the Euclidean distance function for the spatial domain, and we use the Jaccard similarity for the textual set similarity. Other distance functions are supported in a straightforward way. However, extending our work for other text similarity functions is left for future work. Table 1 provides the main symbols used in this paper.

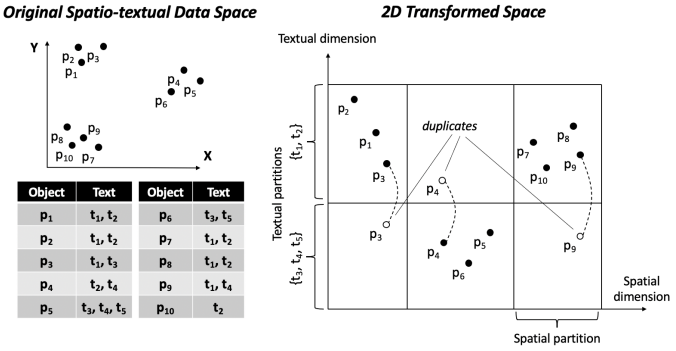


Figure 1: Overview of mapping approach.

Our approach maps the given data objects to data points in a transformed 2D space. The problem is challenging due to the high dimensionality of the original space where data objects are represented, and this is mainly due to the presence of textual information. Figure 1 presents a graphical overview of our approach, with the data set D depicted on the left, whereas the transformed 2D space is shown on the right. In summary, we map the location information

in one dimension (horizontal axis), and the textual information in another dimension (vertical axis). Essentially, in the transformed 2D space, the objects form spatial partitions based on their pairwise distances, as well as textual partitions (in the example $\{t_1, t_2\}$ and $\{t_3, t_4, t_5\}$) based on grouping together subsets of frequently co-occurring keywords. While each object belongs to a single spatial partition, it may be assigned to multiple textual partitions. For example, objects p_1, p_2 and p_3 are assigned to the same spatial partition because they form a spatial cluster in the original space. On the other hand, objects p_3, p_4 and p_9 are duplicated to both textual partitions depicted in Figure 1, because they contain keywords from both textual partitions. The technical details on how this mapping is performed are presented in the next section.

After having obtained the transformed data set, we propose a query processing algorithm that operates on the transformed data and prunes the search space using appropriate bounds, in order to retrieve the exact result set. For this purpose, we follow a *filter-and-refine* approach that operates in the transformed space. For the filtering step, we present a technique that transforms a spatial-keyword range query to *a set of 2D window queries* in the transformed space, in a way that it is guaranteed that the correct result is going to be retrieved if these rectangular areas are searched. In the refinement step, we check whether the data objects present in these areas are truly query results.

Our proposed approach has two main advantages. First, the data in the transformed space can be indexed by any traditional access method designed for 2D and does not need any specialized index for spatio-textual data. Many commercial systems support indexing of 2D data, such as Postgres or MySQL. Secondly, our transformation also partitions the data based on their similarity so that the partitions can be used for parallel query processing in big data frameworks, such as Hadoop or Spark.

In the following, we present the mapping technique for spatio-textual data to 2D values (Section 3), the query processing algorithm (Section 4), as well as an extension (Section 5).

3 DATA MAPPING

Our objective is to transform spatio-textual data objects to data points in a two-dimensional (2D) space, which can be indexed using traditional 2D access methods. To this end, the spatial information and the textual information are mapped to two one-dimensional (1D) values, respectively.

Our approach is based on iDistance [11, 21], an indexing method for similarity search, with main rationale to index the distance between points, instead of their locations. However, iDistance has been proposed for multi-dimensional numeric data and is not straightforward how to generalize for spatio-textual data. In the following, we describe briefly the iDistance mapping for spatial data (Section 3.1) and then we present our adaptation that makes it applicable for textual data (Section 3.2).

3.1 Mapping the Spatial Information

As already mentioned, in order to handle the spatial dimensions, we employ the iDistance technique [11, 21]. Consider a partitioning of the data space into clusters, and each cluster C_i is represented by a reference object K_i (e.g., its centroid) and a radius r_i, which is the distance of the farthest point assigned to C_i from K_i. Any clustering algorithm can be used for the partitioning of the data points. Thus, each point is assigned to the nearest cluster center and mapped to a 1D value according to the distance to its cluster's reference object. The main idea is that 1D values of points that belong to different clusters should be assigned to different 1D ranges, thus a (large enough) constant c is used to separate individual clusters and the iDistance value for an object $p \in C_i$ is defined as:

$$\text{iDist}(p) = i \cdot c + \text{dist}(K_i, p) \quad (1)$$

Expecting that c is large enough, all objects in cluster C_i are mapped to the interval $[i \cdot c, (i+1) \cdot c]$. Additionally to the 1D values, the cluster centroid K_i and the radius r_i of each cluster are maintained, in order to facilitate query processing.

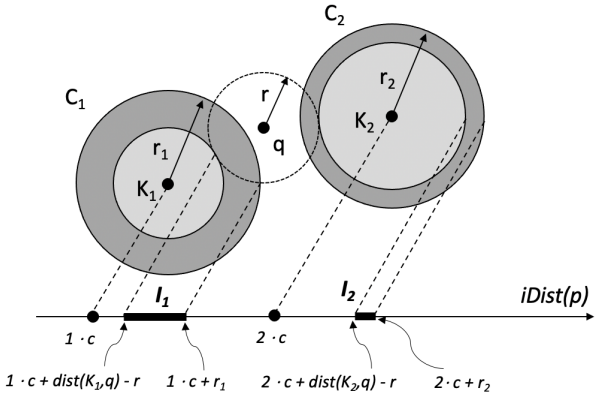


Figure 2: Example of iDistance.

Practically, iDistance transforms the problem of spatial range search to an interval search problem as follows. For a spatial range query, defined by a location q and a radius r, each cluster C_i that satisfies the inequality $\text{dist}(K_i, q) - r \leq r_i$[1] must be visited, as it may contain points within distance r from q. For each such cluster C_i, an interval search is initiated on: $\mathcal{I}_i = [low, high]$, where:

$$\mathcal{I}_i.low = i \cdot c + \min\{\text{dist}(K_i, q) - r, 0\}$$

$$\mathcal{I}_i.high = i \cdot c + \max\{\text{dist}(K_i, q) + r, r_i\}$$

The data points whose iDistance values belong to the interval $\mathcal{I}_i$ are retrieved. Note that these data objects belong to the cluster C_i. For the retrieved points p_i the actual distance to the query point is evaluated and thereafter, if the inequality $\text{dist}(p_i, q) \leq r$ holds, p_i is added to the result set.

Example 3.1. Figure 2 shows a range query and two intersecting clusters C_1 and C_2. The points belonging to cluster C_i are mapped to the interval $[i \cdot c, (i+1) \cdot c]$, using a large enough value of c, so that the respective intervals are disjoint. Since the depicted range query intersects with both C_1 and C_2, two 1D intervals need to be searched. These two intervals $\mathcal{I}_1$ and $\mathcal{I}_2$ are depicted with bold lines on the axis representing the mapped (iDistance) values in the 1D space.

[1]Henceforth also mentioned as intersection of the range query (q, r) and the cluster C_i.

3.2 Mapping the Textual Information

Let V denote the vocabulary of the data set, i.e., $V = \{t_1, t_2, \ldots, t_{|V|}\}$, where t_i represents a keyword of the vocabulary. Our objective is to partition V in k disjoint subsets of keywords. The way to partition V is orthogonal to the proposed approach. A naive approach would be to randomly create k disjoint subsets of V, whose union makes V, i.e., $V_i \cap V_j = \emptyset$ and $\bigcup V_i = V$. Ideally, we would like to partition V in such a way that there exists no data object p in the data set that contains keywords from more than one subset V_i. This is important because if a data object belongs to more than one partitions, then the data object needs to be replicated. In the following, we describe a keyword partitioning approach that minimizes the overlap of keyword sets.

3.2.1 Generating the Textual Partitions. In order to capture the co-occurrence of keywords in spatio-textual objects, we build the *keyword co-occurrence graph*. Then, we apply a graph partitioning algorithm in order to generate the desired subsets of keywords $\{V_1, V_2, \ldots, V_k\}$. These can be considered as the textual partitions of the data set.

The keyword co-occurrence graph is an undirected, weighted graph that contains as vertices the keywords from the vocabulary V. An edge between two keywords (vertices) is added if these keywords co-occur in at least one spatio-textual object. The weight of an edge is set equal to the number of objects in which the respective keywords co-occur. The process of graph construction is of linear complexity $O(n)$ to the number n (= $|D|$) of spatio-textual objects and is provided in Algorithm 1.

Algorithm 1 Keyword co-occurrence graph construction

```
1:  Input: Data set of spatio-textual objects D = {p_1, ..., p_n}
2:  Output: Keyword co-occurrence graph G
3:  for p ∈ D do
4:    for each pair of keywords t_i, t_j in P do
5:      create vertices for keywords t_i and t_j, if they do not exist
6:      if exists edge between vertices t_i, t_j then
7:        increase the weight of edge (t_i, t_j): w_ij = w_ij + 1
8:      else
9:        add edge (t_i, t_j) with weight w_ij = 1
10:     end if
11:   end for
12: end for
13: return G
```

As soon as the graph is constructed, the second step is to invoke a graph partitioning algorithm. Although different algorithms can be applied, we employ METIS [12], which is a widely-used algorithm for graph partitioning. As a result, the vertices of the graph (keywords) are assigned into k groups, in such a way that keywords that frequently co-occur together are placed in the same group. Consequently, we obtain k disjoint partitions $\{V_1, V_2, \ldots, V_k\}$ of the vocabulary V, where each partition V_i contains a subset of the keywords of vocabulary V.

3.2.2 Mapping the Keywords of a Spatio-textual Object. A spatio-textual data object p may contain keywords from different (say $\ell \geq 1$) subsets V_i. Thus, in contrast to the spatial iDistance, we need to assign such a spatio-textual object p to ℓ partitions. Therefore, p is replicated ℓ times in the transformed data set. This is necessary in order to ensure the correctness of the computed result set for any spatial-keyword range query.

For a given spatio-textual data object p (with keyword set P) and a partition V_i for which it holds that $P \cap V_i \neq \emptyset$, we define a 1D similarity value $\text{val}(P, V_i)$ based on the following equation:

$$\text{val}(P, V_i) = \frac{|P \cap V_i|}{|P|} \tag{2}$$

which practically "distributes" the size of the overlap between sets P and the vocabulary V to those partitions V_i that have $P \cap V_i \neq \emptyset$. In fact, $\text{val}(P, V_i)$ represents the size of the overlap normalized over the size of P.

Example 3.2. Consider a vocabulary $V = \{t_1, t_2, \ldots, t_9\}$ and let us assume ℓ=2 disjoint partitions: $V_1 = \{t_1, \ldots, t_5\}$ and $V_2 = \{t_6, \ldots, t_9\}$. For an object p with $P = \{t_1, t_4, t_6, t_7, t_8\}$, its similarity value based on V_1 is $\text{val}(P, V_1) = \frac{|P \cap V_1|}{|P|} = \frac{2}{5}$. Similarly, it holds that $\text{val}(P, V_2) = \frac{|P \cap V_2|}{|P|} = \frac{3}{5}$. In this way, we distribute the normalized overlap of p to the two partitions, according to the overlap of keyword set P with the keywords in the partitions V_1 and V_2.

The next step is to assign similarity values of data points for different partitions V_i,V_j to disjoint 1D intervals of the textual dimension. Thus, the textual similarity values $\text{val}(P, V_i)$ are mapped to 1D values in such a way that only data objects that have keywords that belong to the partition V_i are mapped to the same interval. Expecting that c' is large enough, similar to the concept of the spatial iDistance, all data objects p with keywords belonging to the partition V_i ($P \cap V_i \neq \emptyset$) are mapped to the 1D values:

$$\text{iSim}(p, V_i) = i \cdot c' + \text{val}(P, V_i)$$

The remaining question is how to bound the value $\text{val}(P, V_i)$. Put differently, how to determine the intervals that correspond to data objects p for which it holds that $\text{sim}(P, Q) \geq \tau$ for a given query q. Given a query q and a keyword partition V_i with overlapping keywords to q, i.e., $Q \cap V_i \neq \emptyset$, for a data object p it holds that $P \cap V_i \neq \emptyset$ and $\text{sim}(P, Q) \geq \tau$, if $\text{val}(P, V_i)$ is within the interval $\mathcal{J}_i = [low, high]$, where:

$$\mathcal{J}_i.low = i \cdot c' + \text{minscore}_i$$

$$\mathcal{J}_i.high = i \cdot c' + \text{maxscore}_i$$

while minscore_i and maxscore_i are defined in the following lemma.

Lemma 3.3. *Given a query q and a keyword partition V_i, such that $Q \cap V_i \neq \emptyset$, for any data object p for which* $\text{sim}(P, Q) \geq \tau$ *and $P \cap V_i \neq \emptyset$, it holds that* $\text{val}(P, V_i) \in [\text{minscore}_i, \text{maxscore}_i]$, *where:*

$$\text{minscore}_i = \tau - \frac{|Q \cap (V - V_i)|}{|Q|} \tag{3}$$

$$\text{maxscore}_i = \frac{|Q \cap V_i|}{|Q|} + 1 - \tau \tag{4}$$

Proof. The intersection $|P \cap Q|$ of keyword sets P and Q can be split in two parts. The first part represents the keywords that also belong to partition V_i, while the second part represents the remaining keywords:

$|P \cap Q| = |P \cap Q \cap (V_i \cup (V - V_i))| = |(P \cap Q \cap V_i) \cup (P \cap Q \cap (V - V_i))| =$

$$|P \cap Q \cap V_i| + |P \cap Q \cap (V - V_i)|$$

Thus, we can write $\text{sim}(P, Q)$ as follows:

$$\text{sim}(P, Q) = \frac{|P \cap Q|}{|P \cup Q|} = \frac{|P \cap Q \cap V_i|}{|P \cup Q|} + \frac{|P \cap Q \cap (V - V_i)|}{|P \cup Q|} \quad (5)$$

Lower bound: Because it holds that: $|P \cap Q \cap V_i| \leq |P \cap V_i|$ and $|P \cup Q| \geq |P|$, we derive from Eq. 5:

$$\text{sim}(P, Q) = \frac{|P \cap Q \cap V_i|}{|P \cup Q|} + \frac{|P \cap Q \cap (V - V_i)|}{|P \cup Q|} \leq$$

$$\frac{|P \cap V_i|}{|P|} + \frac{|P \cap Q \cap (V - V_i)|}{|P \cup Q|} \leq$$

$$\text{val}(P, V_i) + \frac{|P \cap Q \cap (V - V_i)|}{|P \cup Q|}$$

because: $\text{val}(P, V_i) = \frac{|P \cap V_i|}{|P|}$ (by definition). Similarly, since: $|P \cap Q \cap (V - V_i)| \leq |Q \cap (V - V_i)|$ and $|P \cup Q| \geq |Q|$, we further derive that:

$$\text{sim}(P, Q) \leq \text{val}(P, V_i) + \frac{|Q \cap (V - V_i)|}{|Q|}$$

Thus, we can rewrite the latter inequality as follows:

$$\text{val}(P, V_i) \geq \text{sim}(P, Q) - \frac{|Q \cap (V - V_i)|}{|Q|}$$

which eventually leads to:

$$\text{val}(p, V_i) \geq \tau - \frac{|Q \cap (V - V_i)|}{|Q|} = \text{minscore}_i$$

Upper bound: Because it holds that $|P \cap Q \cap V_i| \leq |Q \cap V_i|$ and $|P \cup Q| \geq |Q|$, we derive from Eq. 5:

$$\text{sim}(P, Q) = \frac{|P \cap Q \cap V_i|}{|P \cup Q|} + \frac{|P \cap Q \cap (V - V_i)|}{|P \cup Q|} \leq$$

$$\frac{|Q \cap V_i|}{|Q|} + \frac{|P \cap Q \cap (V - V_i)|}{|P \cup Q|}$$

Furthermore, since it holds that: $|P \cap Q \cap (V - V_i)| \leq |P \cap (V - V_i)|$ and $|P \cup Q| \geq |P|$, we have:

$$\text{sim}(P, Q) \leq \frac{|Q \cap V_i|}{|Q|} + \frac{|P \cap (V - V_i)|}{|P|} =$$

$$\frac{|Q \cap V_i|}{|Q|} + \frac{|P \cap V|}{|P|} - \frac{|P \cap V_i|}{|P|}$$

Moreover, $|P \cap V| \leq |P|$ and $\text{val}(P, V_i) = \frac{|P \cap V_i|}{|P|}$ (by definition), we derive:

$$\text{sim}(P, Q) \leq \frac{|Q \cap V_i|}{|Q|} + 1 - \frac{|P \cap V_i|}{|P|} = \frac{|Q \cap V_i|}{|Q|} + 1 - \text{val}(P, V_i)$$

Thus, we can rewrite the latter inequality: $\frac{|Q \cap V_i|}{|Q|} + 1 - \text{val}(P, V_i) \geq \tau$, which leads to:

$$\text{val}(P, V_i) \leq \frac{|Q \cap V_i|}{|Q|} + 1 - \tau = \text{maxscore}_i$$

□

The consequence of this Lemma is that for a given query q with query keywords Q: (a) we only need to search in those textual partitions V_i that have common keywords with Q (i.e., $Q \cap V_i \neq \emptyset$), and (b) for each such textual partition V_i, we need to search only within the bounds of Eq. 3 and 4 defined in Lemma 3.3.

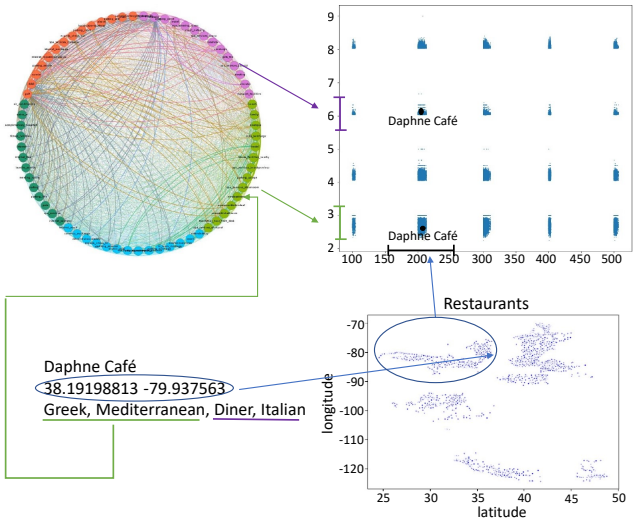


Figure 3: Mapping spatio-textual objects to a two-dimensional (2D) space.

Example 3.4. Figure 3 illustrates the overall approach graphically. At the bottom right, the spatial distribution of the data set is depicted in the geographical 2D space. At the left, a spatio-textual object p, called "Daphne Café", is shown with its coordinates $(p.x, p.y)$ and its textual description P={Greek, Mediterranean, Diner, Italian}. At the top left, the keyword co-occurrence graph is depicted, where textual partitions are depicted using different colors. Finally, at the top right, the transformed 2D data space is shown, and the spatio-textual objects are mapped to regions of the 2D space. Notice the arrows that indicate how textual partitions (on the vertical axis) and spatial clusters (on the horizontal axis) correspond to 1D intervals of values. Also, note that object p is mapped to two points in the transformed space, because its keywords have been assigned to two distinct textual partitions (denoted with purple and green colours respectively).

4 QUERY PROCESSING

We assume that the spatio-textual data objects in data set D are transformed to a 2D space (as described earlier) and indexed by a traditional spatial index (such as an R-tree) that supports range queries.

Algorithm 2 describes the *ST2D* algorithm for spatial-keyword range query processing in the transformed 2D space. Our approach adheres to the *filter-and-refinement* methodology. In particular, the filtering phase is described in lines 4–8, while the refinement phase corresponds to lines 9–13. In the following, we explain the two phases in more detail.

Filtering in the Transformed Space. A spatial-keyword range query q, defined by location $(q.x, q.y)$, query keywords Q, as well as distance threshold r and textual similarity threshold τ, is transformed to a set of window queries in the transformed 2D space.

With respect to the spatial dimension of the transformed space, for each cluster $C_i(K_i, r_i)$ that intersects with the circle centered at $(q.x, q.y)$ and radius r defined by the spatial part of the query, we

Algorithm 2 *ST2D*: Spatial-keyword range algorithm in transformed space

1: **Input:** Query q, radius r, similarity threshold τ
2: **Output:** Result set $\mathcal{R}$
3: $\mathcal{R} \leftarrow \emptyset$, $cand \leftarrow \emptyset$
4: $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\} \leftarrow$ 1D intervals for spatial dimension
5: $\mathcal{J} = \{\mathcal{J}_1, \ldots, \mathcal{J}_m\} \leftarrow$ 1D intervals for textual dimension
6: **for** (each pair $(\mathcal{I}_i, \mathcal{J}_j)$) **do**
7: $cand \leftarrow cand \cup$ WindowQuery$(\mathcal{I}_i, \mathcal{J}_j)$ {***Filter step***}
8: **end for**
9: **for** ($p \in cand$) **do**
10: **if** (dist$(p, q) \leq r$ and sim$(P, Q) \geq \tau$) **then**
11: $\mathcal{R} \leftarrow \mathcal{R} \cup p$ {***Refinement step***}
12: **end if**
13: **end for**
14: **return** $\mathcal{R}$

need to retrieve the data objects with iDistance values belonging to the interval $\mathcal{I}_i$ as defined in Section 3.1. Obviously, when the query intersects more than one clusters, we obtain a set of intervals $\{\mathcal{I}_1, \mathcal{I}_2, \ldots\}$, as many as the intersecting clusters with the query (line 4).

With respect to the textual dimension of the transformed space, for each partition V_i that has at least one common term with the query keywords Q, i.e., $V_i \cap Q \neq \emptyset$, we obtain an interval that needs to be searched. For partition V_i, the interval $\mathcal{J}_i$ is obtained, as described in Section 3.2. Again, multiple such intervals $\{\mathcal{J}_1, \mathcal{J}_2, \ldots\}$ are defined, as many as the partitions V_i that have at least one common term with the query keywords (line 5).

Finally, assuming n intervals $\{\mathcal{I}_i\}$ and m intervals $\{\mathcal{J}_i\}$, the spatial-keyword range query is equivalent to $n \cdot m$ window queries in the transformed space. These queries correspond to the rectangles $[\mathcal{I}_i, \mathcal{J}_j,]$ for $i \in [1, n]$ and $j \in [1, m]$, and can be efficiently processed by exploiting a traditional 2D index, such as an R-tree (lines 6–8).

Refinement Phase. All data objects retrieved by the window queries are *candidate* objects for the result set $\mathcal{R}$. From the candidate results $p \in cand$, only those satisfying the inequality dist$(p, q) \leq r$ are results with respect to the spatial constraint. Also, the candidate object must satisfy the textual similarity constraint, i.e., sim$(P, Q) \geq \tau$. A candidate object is a result of the spatial-keyword range query, only if it satisfies both constraints, otherwise it is dismissed as false positive. This is checked in the refinement phase of the algorithm (lines 9–13).

On a final note, recall that an object may be duplicated in the transformed 2D space due to the fact that its keywords match different textual partitions. Thus, during query processing, when an object is retrieved from window query, it is processed further only if it has not already been retrieved from another window query, otherwise it is immediately discarded. This is checked whenever we add an object to the candidate result set.

5 PROXIMITY-AWARE PARTITION RE-ORDERING

As already mentioned, *ST2D* provides an efficient solution to the problem of spatial-keyword range query processing by mapping the spatial and textual dimension to a transformed 2D space and employing a *filter-and-refinement* methodology. However, a possible bottleneck that might affect the efficiency of this solution when implemented inside a DBMS, is the multiple $(n \cdot m)$ index look-ups (lines 4-8, Algorithm 2) that take place due to the different 1D intervals for the spatial and textual dimension.

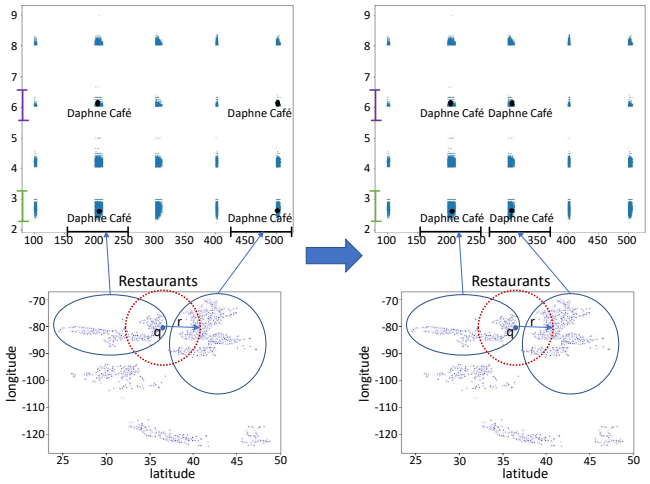


Figure 4: Proximity-aware spatial partition re-ordering.

Assuming n intervals $\{\mathcal{I}_i\}$ for the spatial dimension, and m intervals $\{\mathcal{J}_i\}$ for the textual dimension, a straightforward way to deal with this is to get the minimum ($min(\mathcal{I}_i.low)$ and $min(\mathcal{J}_j.low)$) and maximum ($max(\mathcal{I}_i.high)$ and $max(\mathcal{J}_j.high)$) bounds of intervals $\mathcal{I}$ and $\mathcal{J}$ and create a new pair of intervals $\mathcal{B} = (\mathcal{I}_i, \mathcal{J}_j)$ that corresponds to the minimum bounding rectangle of the initial rectangles $[\mathcal{I}_i, \mathcal{J}_j,]$ for $i \in [1, n]$ and $j \in [1, m]$. Nevertheless, as depicted in Figure 4 (left), the different intervals for each dimension might not be continuous or even close to each other. Consequently, utilizing $\mathcal{B}$ to perform the filtering will lead to having a large number of "irrelevant" data returned by the filtering phase (false positives), which in turn will lead to a more "expensive" refinement phase.

Algorithm 3 *PPR_ST2D*: Spatial-keyword range algorithm in transformed space with proximity-aware parition re-ordering

1: **Input:** Query q, radius r, similarity threshold τ
2: **Output:** Result set $\mathcal{R}$
3: $\mathcal{R} \leftarrow \emptyset$, $cand \leftarrow \emptyset$
4: $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\} \leftarrow$ 1D intervals for spatial dimension
5: $\mathcal{J} = \{\mathcal{J}_1, \ldots, \mathcal{J}_m\} \leftarrow$ 1D intervals for textual dimension
6: **calculate** $\mathcal{B}$ of $\mathcal{I}$ and $\mathcal{J}$
7: $cand \leftarrow$ WindowQuery$(\mathcal{B})$ {***Filter step***}
8: **for** ($p \in cand$) **do**
9: **if** (dist$(p, q) \leq r$ and sim$(P, Q) \geq \tau$) **then**
10: $\mathcal{R} \leftarrow \mathcal{R} \cup p$ {***Refinement step***}
11: **end if**
12: **end for**
13: **return** $\mathcal{R}$

For this reason, we propose the proximity-aware partition reordering solution, where the goal is to order spatial partitions in such a way that partitions that are "close" to each other in the original 2D Euclidean space, will also be close to the transformed iDistance-based 1D space, as illustrated in Figure 4 (right). Based on Equation 1, this can be regulated by the i parameter, which serves as the spatial cluster identifier (ID). Hence, we need to re-order clusters IDs in such a way so that clusters that have spatial proximity will have consecutive or at least close IDs. To achieve this, we utilize a space-filling curve, and more specifically z-order (however other space-filling curves might also be applied), on each reference object K_i of the clusters. Subsequently, we order the clusters by the z-order of their reference points and re-assign the cluster ids i based on this order, where $i \in [1, |C|]$ and $|C|$ is the number of spatial clusters.

By doing so, as described in Algorithm 3 (*PPR_ST2D*), during the *filtering* phase we perform a single look-up in the spatial index (line 7), instead of $n \cdot m$, where n is the number of 1D intervals for the spatial dimension and m the 1D intervals for the textual dimension. *PPR_ST2D* exploits the proximity-aware partition reordering so as to reduce the number of irrelevant data objects retrieved. The refinement procedure remains the same (lines 8-12).

6 EXPERIMENTAL EVALUATION

In this section, we present the results of the experimental evaluation of our approach. All the experiments were conducted in an Intel(R) Core(TM) i7-8750H CPU (2.20GHz), with 16GM of RAM and 1TB of disk. The mapping algorithms were implemented in Java, while the actual range query was implemented in PL/SQL. For our experimental study we utilized PostgreSQL 13 and PostGIS, with the default settings. Technically, the data objects are stored in a table and the queries are implemented in PL/SQL. The R-tree spatial index was implemented inside PostgreSQL by utilizing the GiST inteface, while the inverted indexes using the GIN interface.

6.1 Experimental Setup

For our experimental study, we employed two real data sets of spatio-textual objects, namely *Booking* and *Factual*. The former data set, is a set of 200,000 descriptions of hotels crawled from the site of Booking.com[2]. The data set contains $|V| = 188$ distinct keywords. The average, minimum and maximum number of keywords per object in the *Booking* data set are 18, 0, 106 respectively. The latter data set, is a set of 104,444 descriptions of restaurants and hotels located in North America, crawled from Factual[3]. The data set contains $|V| = 199$ distinct keywords. The average, minimum and maximum number of keywords per object in the data set D are 10, 1, 47 respectively.

Algorithms. For our experimental study we are going to utilize 2 baseline solutions to the problem of spatial-keyword range query (*spatial_first* and *best_first*), the solution proposed in [15] (*STbHI*), the solution presented in Section 4 (*ST2D*) and the improved solution presented in Section 5 (*PPR_ST2D*).

Concerning *spatial_first*, we initially filter spatially the data set, by employing the R-tree index provided by PostGIS and for the resulting spatio-textual points we verify the textual predicate. Regarding *best_first*, in addition to the R-tree spatial index we built an inverted index on the keyword set and utilize it in order to filter out spatio-textual objects whose keyword sets do not intersect with the keyword set of q. Furthermore, we let the DBMS select the order that the predicates (spatial and textual) will be evaluated.

Also, the solution proposed in [15], *STbHI*, maps the spatial information to 1D using a space-filling curve (z-order). Then, each keyword gets concatenated with this 1D value and an inverted index is built on this set, thus duplicating objects as many times as the number of keywords. This approach favors the textual dimension during query processing, as the keyword precedes the 1D value in the string representation. Then, for each spatio-textual query point q, the spatio-textual objects of D that contain at least one keyword are retrieved, by utilizing the inverted index and then the spatial predicate gets evaluated by mapping the location of q in 1D, using the same space-filling curve, and translating the spatial query into a range query on the z-order values.

Parameter	Values
Data set size $\|D\|$	x2, x4, x6, x8, x10
Query radius r (in km)	0, 2, 4, **6**, 8, 10
Textual similarity threshold τ	0, 0.2, 0.4, **0.6**, 0.8, 1
# of query keywords $\|Q\|$	1, 2, 3, 4, 5
# of spatial partitions $\|C\|$ (in thousands)	0.5, 1, 1.5, **2**, 2.5, 3
# of textual partitions k	5, 10, 15, **20**, 25, 30

Table 2: Parameter values (default in bold).

Methodology. Initially, we evaluate the scalability of the proposed solutions by increasing the data set size and measure the execution time. Subsequently, we compare the performance of the solutions that we employed for our experimental study. Successively, we investigate the effect of setting different values to r and τ to the execution time. Next, we examine how the number of query keywords ($|Q|$) affects the performance of the proposed solutions. Finally, we investigate the effect of the number of spatial partitions $|C|$ and the number of textual partitions k to the performance of *ST2D* and *PPR_ST2D*. The different parameter values are depicted in Table 2, while their default setting is depicted in bold.

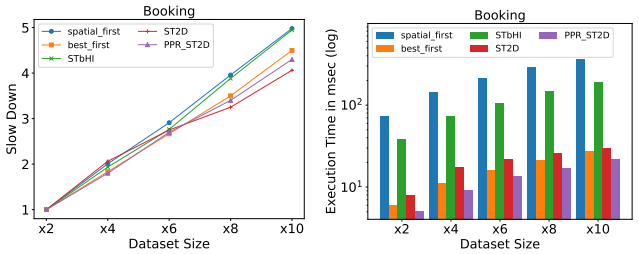
Queries. For each experiment we perform 200 queries and use the median execution time. Each query is generated by randomly selecting an already existing spatio-textual point from the respective data set. In the experiment where we vary the number of keywords, we first filter the data set and keep only the spatio-textual points that have a specific number of keywords and then we randomly select the ones that are used. The query radius r and the similarity threshold τ are set based on the parameters of the experiment, as depicted in Table 2.

6.2 Results

6.2.1 Scalability. As already mentioned, initially, we vary the size of our data set and measure the execution time of the algorithms. To study the effect of data set size, we focused on the *Booking* data set, which is almost double in size than *Factual*, and we created 5 portions (x2, x4, x6, x8, x10) of the original data set of increased sizes

[2]http://www.booking.com
[3]https://www.factual.com/

up to 2M objects. As the data set size increases, it is expected that the execution time will increase too. In order to measure the factor of this increase, for each portion D_i of the data set with $i \in [1, 5]$, we calculate $SlowDown = \frac{T_{D_i}}{T_{D_1}}$, where T_{D_1} is the execution time of the first portion (i.e. x2) and T_{D_i} the execution time of the current one. As illustrated in Figure 5(a), all of the solutions scale linearly with the size of the data set. However, we can observe two distinct groups based on scalability, a group consisting of *spatial_first* and *STbHI*, that appear to have linear scalability and another group that consists of *best_first*, *ST2D* and *PPR_ST2D*, that seem to exhibit a slightly better performance in terms of scalability.



(a) (b)

Figure 5: Scalability when varying the data set size (a) in terms of *SlowDown* and (b) Execution time.

Furthermore, as depicted in Figure 5(b), it can be observed that *spatial_first* performs worse than the other 4 approaches and by almost an order of magnitude slower than *ST2D* and *PPR_ST2D*. This behavior is somewhat expected since in *best_first* we utilize both a spatial and a textual index. Moreover, we allow the DBMS to utilize the query optimizer and choose the evaluation order of the predicates. Furthermore, *STbHI* prioritizes the textual part of the query, which has better selectivity than the spatial part of the query since in the *Booking* data set we have 188 distinct keywords. This also supported by the examining the extreme cases where $r = 0$ and $\tau = 0$, as depicted in Figures 6(a) and (c), where in the former *spatial_first* outperforms *STbHI* and in the latter is the other way around. Furthermore, in *ST2D* and *PPR_ST2D* the spatial and textual information are mapped in 2D space and then the R-tree spatial index is utilized for both the spatial and textual dimension. For the same reasons, *best_first*, *ST2D* and *PPR_ST2D* outperform *STbHI*, since in *best_first* the order of predicates is selected by the query optimizer and in *ST2D* and *PPR_ST2D* the spatial and textual information are treated equally. In addition, it can be observed that *ST2D* performs worse than *best_first* and *PPR_ST2D* mostly due to the fact that for each query the we need to perform multiple look-ups on the spatial index, as described in Algorithm 2. Finally, we can see that *PPR_ST2D* outperforms all of the other approaches since it filters more efficiently the data set by employing the data mapping technique presented in Section 3 and on top of this it performs one look-up per query, which speeds up significantly the filtering phase of the query.

6.2.2 Sensitivity Analysis. Subsequently, we study the effect of the spatial predicate on the execution time. Towards this direction, we set different values to r while keeping the values of the other parameters fixed. In more detail, as delineated in Figures 6(a) and (d), the larger the r, the higher the execution time of the query for all 5 approaches. Next, we vary the values of the textual predicate τ while keeping the values of the other parameters fixed. As presented in Figures 6(b) and (e), the *spatial_first*, *best_first* and *STbHI* solutions, does not seem to be affected by τ. On the other hand, we can observe a decreasing trend in *ST2D* and *PPR_ST2D* as τ increases. This behaviour is caused by the fact that both of these solutions utilize the textual information more efficiently at the filtering stage of the query, since the adopt the mapping technique presented in Section 3. Hence, the more "strict" the query becomes, as far as it concerns the textual information, the less data qualify for the filtering step.

Concerning the special case where $r = 0$, we can observe that the *spatial_first* solution performs equally well as the *PPR_ST2D*, since it initially performs the spatial filtering by utilizing the R-Tree index, which in turn leads to a very "light" refinement phase. On the other hand, *ST2D* does not perform very well due to the large number of look-ups in the index. Moreover, *best_first* performs slightly worse than *spatial_first*, since it depends on the query optimizer to select the order of predicates, while *spatial_first* always chooses to filter the spatial dimension first. Finally, *STbHI* performs worse than *spatial_first*, *best_first* and *PPR_ST2D*, since it prioritizes the textual dimension. Considering the special case where $\tau = 0$, we observe that the *spatial_first* solution performs worse, since it does not filter the search space efficiently. Moreover, even though *STbHI* prioritizes the textual dimension, it does not incorporate into the search the Jaccard similarity, hence it first retrieves all the records that contain the specific keywords and then evaluate their Jaccard similarity. On the contrary, *ST2D* and *PPR_ST2D* perform a lot better since they both incorporate the Jaccard metric in the search, by adjusting accordingly the upper and lower bound of the search range of the textual dimension in the transformed 2D space.

Moreover, we examine how the number of query keywords affects the execution time. As illustrated in Figures 6(c) and (f), it appears that there is an increasing trend in the execution time as the number of query keywords increase for all 5 approaches.

Finally, we investigate the effect of the number of spatial and textual partitions to the execution time of *ST2D* and *PPR_ST2D*. Regarding the number of spatial partitions, as illustrated in Figure 7(a), it does not seem to affect either the *ST2D* or the *PPR_ST2D* solutions. On the other hand, concerning the number of textual partitions, we can observe an obvious increasing trend in the execution time as the number of textual partitions increase to both *ST2D* or the *PPR_ST2D*, as depicted in Figure 7(b). This is due to the fact that, as the number of textual partitions increase, the amount of replicated data will also increase, since each spatio-textual data object can be assigned to multiple textual partitions, as explained in Section 3.2.2.

7 RELATED WORK

Due to the popularity of spatial-keyword queries and their wide applicability in practical scenarios, several approaches have been proposed to support efficient query processing [3, 4, 6, 17].

There exist some approaches for mapping spatio-textual data to lower dimensional representations, in particular 1D alphanumeric

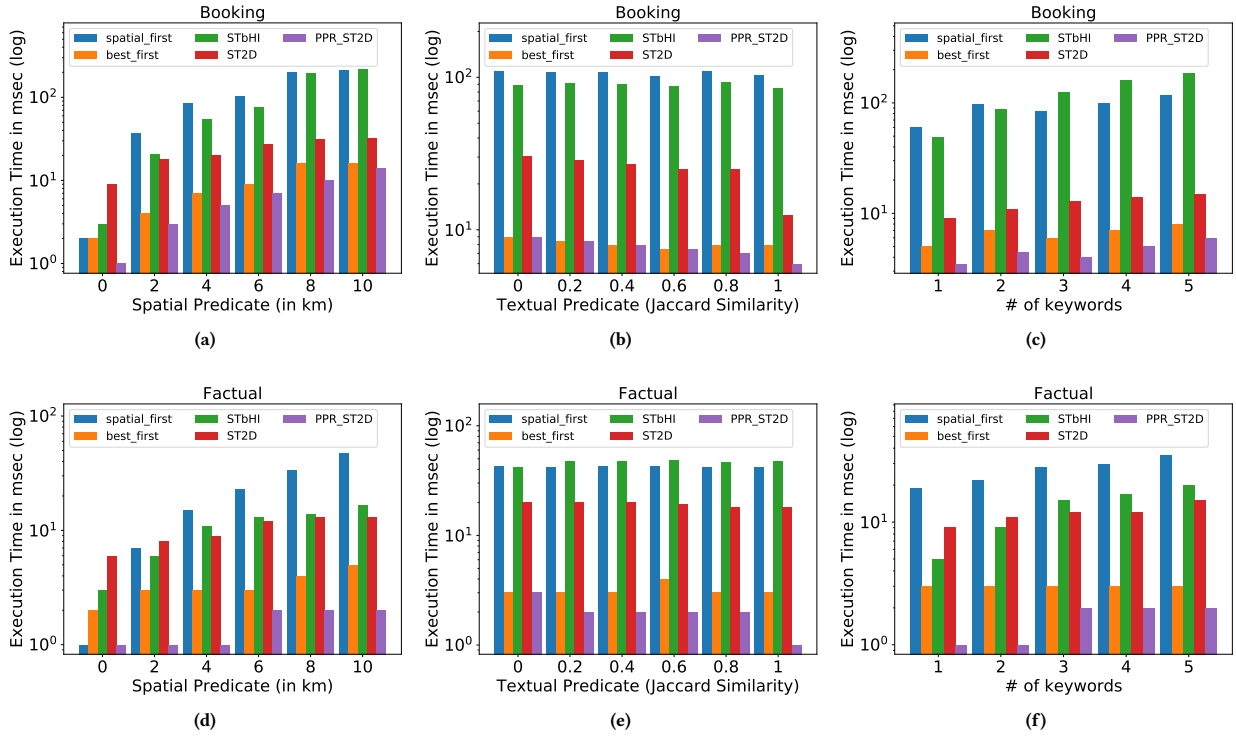


Figure 6: Execution time when varying the spatial predicate on (a) the *Booking* and (d) the *Factual* data set, the textual predicate on (b) the *Booking* and (e) the *Factual* data set and the number of query keywords on (c) the *Booking* and (f) the *Factual* data set.

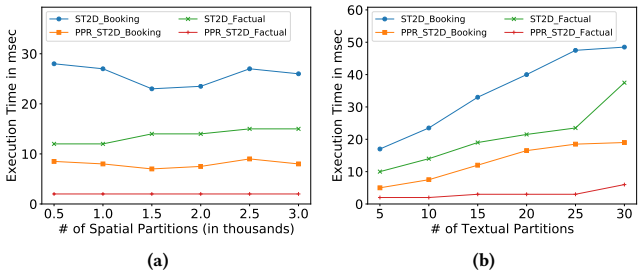


Figure 7: Execution time when varying the number of (a) Spatial and (b) Textual partitions.

values, which can be used as keys by data access mechanisms that support key-based retrieval. For example, ST-HBase [15] maps the spatial information to 1D using a space-filling curve and concatenate each keyword with the 1D value, thus duplicating objects as many times as the number of keywords. However, this 1D mapping inevitably favors the textual dimension during query processing, as the keyword precedes the 1D value in the string representation. Differently, in our work which is performed in the context of the SPADES project [19], we propose a mapping that transforms spatio-textual data to a 2D space, where the dimensions of space and text are of equal importance.

The most prominent approaches for effective indexing of spatio-textual data rely on a specialized access method for joint management of spatial and textual dimensions, so as to facilitate efficient query processing. In general, the approaches are classified as spatial-first, text-first or interleaved, based on the primary dimension (space or text) selected to organize the data [17]. Early approaches [5, 7] have studied the benefits of spatial-first vs. text-first indexing.

In [10], the spatial-keyword query is defined as a combination of range queries and boolean keyword search. A special case is the distance-first top-k spatial keyword query [9], which returns a ranked list of the k objects that contain all of keywords and are closest to the query location. That is, distance-first top-k spatial keyword query is a combination of a top-k spatial query and a boolean keyword query. In [9] the IR^2-Tree was proposed which is a combination of R-Tree and the signature file. Two different [8, 14] indexing approaches have been proposed that employ a hybrid index that augments the nodes of an R-tree with inverted indexes. The inverted index at each node refers to a pseudo-document that represents all the objects under the node. Another hybrid indexing structure that combines the R*-tree and bitmap indexing to process the spatial-keyword query was proposed in [22].

The Spatial Inverted Index (S2I) was proposed in [18] for processing top-k spatial keyword queries more efficiently. The S2I index maps each keyword to a different aggregate R-tree that stores the objects with the given term. The aggregate R-tree stores the latitude and longitude of the objects, and maintains an aggregated value that represents the maximum term impact of the objects under the node. In fact, the aggregated R-tree is employed only when the number of objects exceeds a given threshold. Otherwise, the objects are stored in a file, one block per term.

Interleaved indexing approaches have also been studied in [16, 20]. The AP-tree [20] combines a Quadtree with a trie built on keywords into a single data structure. Since both Quadtrees and tries are hierarchical structures, the AP-tree is also a hierarchical structure with the distinguishing feature that a node can be either spatial or text node. FAST [16] is an approach that integrates the spatial pyramid [2] with a new text index, called Adaptive Keyword Index (AKI). AKI is a hybrid data structure for text, which includes features from inverted lists and keyword tries. In its basic form, AKI keeps inverted lists for each keyword present in the collection. However, it has been observed that when the length of posting lists increases too much, the performance of query processing deteriorates. Motivated by this fact, AKI identifies such long postings and changes their structure to resemble a trie, thus using more keywords to distinguish the objects.

Even though the afore-described methods for efficient querying spatio-textual data are effective, they typically rely on a specialized indexing structure, which has high memory requirements. Instead, our approach does not require a specialized index for spatio-textual queries, and also enables effective partitioning of the data that preserves data locality. Finally, with respect to spatio-textual query types, the query targeted in our work is an approximate keyword range query (similar to [1]), where the keyword constraint is not boolean. Instead, we retrieve the objects that are more similar to the query keywords, based on Jaccard similarity, and belong to a spatial query range.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel approach of transforming spatio-textual data to a two-dimensional (2D) data space. The first axis represent the spatial information and we employ the iDistance concept for computing the value of this dimension for each data object. The second axis represents the textual information. First, a graph is created that captures the co-occurrence of the keywords and then a graph partitioning algorithm is employed for creating disjointed textual partitions. We provide a novel mapping of the data objects to an one-dimensional (1d) value based on the textual partitioning and provide a novel bounding schema to avoid accessing all objects during query processing. Finally, in our experiments, we demonstrate the efficiency of our proposed approach. Regarding future work, we intend to study generalizations of the proposed approach, for different spatial-keyword query types as well as for other text similarity functions.

Acknowledgements

This work was supported from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under grant agreement No 1667, and under the "First Call for HFRI Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-81).

REFERENCES

[1] Sattam Alsubaiee, Alexander Behm, and Chen Li. 2010. Supporting location-based approximate-keyword queries. In *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, November 3-5, 2010, San Jose, CA, USA, Proceedings*. ACM, 61–70.

[2] Walid G. Aref and Hanan Samet. 1990. Efficient Processing of Window Queries in The Pyramid Data Structure. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*. ACM Press, 265–272.

[3] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial Keyword Query Processing: An Experimental Evaluation. *PVLDB* 6, 3 (2013), 217–228.

[4] Lisi Chen, Shuo Shang, Chengcheng Yang, and Jing Li. 2020. Spatial keyword search: a survey. *GeoInformatica* 24, 1 (2020), 85–106.

[5] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. 2006. Efficient query processing in geographic web search engines. In *SIGMOD Conference*. 277–288.

[6] Zhida Chen, Lisi Chen, Gao Cong, and Christian S. Jensen. 2021. Location- and keyword-based querying of geo-textual data: A survey. *VLDB Journal* (2021).

[7] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. 2011. Text vs. space: efficient geo-search query processing. In *CIKM*. 423–432.

[8] Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *PVLDB* 2, 1 (2009), 337–348.

[9] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword Search on Spatial Databases. In *ICDE*. 656–665.

[10] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. 2007. Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems. In *SSDBM*. 16.

[11] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems* 30, 2 (June 2005), 364–397.

[12] George Karypis and Vipin Kumar. 1997. METIS—A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices. (01 1997).

[13] Taesung Lee, Jin-Woo Park, Sanghoon Lee, Seung-won Hwang, Sameh Elnikety, and Yuxiong He. 2015. Processing and Optimizing Main Memory Spatial-Keyword Queries. *Proc. VLDB Endow.* 9, 3 (2015), 132–143.

[14] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. 2011. IR-Tree: An Efficient Index for Geographic Document Search. *IEEE Trans. Knowl. Data Eng.* 23, 4 (2011), 585–599.

[15] Youzhong Ma, Yu Zhang, and Xiaofeng Meng. 2013. ST-HBase: A Scalable Data Management System for Massive Geo-tagged Objects. In *Web-Age Information Management - 14th International Conference, WAIM 2013, Beidaihe, China, June 14-16, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7923)*. Springer, 155–166.

[16] Ahmed R. Mahmood, Ahmed M. Aly, and Walid G. Aref. 2018. FAST: Frequency-Aware Indexing for Spatio-Textual Data Streams. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 305–316.

[17] Ahmed R. Mahmood and Walid G. Aref. 2019. *Scalable Processing of Spatial-Keyword Queries*. Morgan & Claypool Publishers.

[18] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. 2011. Efficient Processing of Top-k Spatial Keyword Queries. In *SSTD*. 205–222.

[19] Akrivi Vlachou, Christos Doulkeridis, Nikolaos Koutroumanis, Dimitrios Poulopoulos, and Kjetil Nørvåg. 2020. The SPADES Framework for Scalable Management of Spatio-textual Data. In *Proceedings of 24th Pan-Hellenic Conference on Informatics (PCI'20)*. ACM.

[20] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2015. AP-Tree: Efficiently support continuous spatial-keyword queries over stream. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 1107–1118.

[21] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. 2001. Indexing the Distance: An Efficient Method to KNN Processing. In *Proceedings of VLDB'01*.

[22] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. 2009. Keyword Search in Spatial Databases: Towards Searching by Document. In *ICDE*. 688–699.