Spatially Combined Keyword Searches

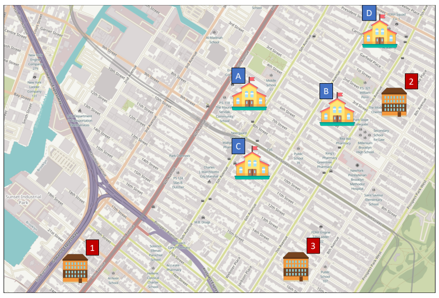
Artur Titkov
Johannes Gutenberg University Mainz, Germany
artitkov@uni-mainz.de

Panagiotis Bouros
Johannes Gutenberg University Mainz, Germany
bouros@uni-mainz.de

ABSTRACT

Data have become increasingly more complex over the past years; entities are easily 'tagged' with different types of auxiliary information, such as text and spatial locations. Examples include geo-tagged news and micro-blogs, map and social services etc. Spatial keyword search is a popular retrieval task in such collections, where objects are reported by their distance to the query location and the relevance of their text to the query keywords. However, the case of spatially combining multiple keyword searches was never studied. Under this prism, we introduce two novel retrieval tasks that either spatially join (SpaJKS) or determine the closest pairs among (SpaRKS) the results of independent keyword searches. To efficiently compute SpaJKS and SpaRKS queries, we also present five strategies employing different types of indexing. Last, our experimental analysis evaluates the performance of these strategies on both real and synthetic spatio-textual objects.

$dist(1, B) = 1.8km$
$dist(1, C) = 1.2km$
$dist(2, B) = 300m$
$dist(2, C) = 900m$



apartment	amenities
1	*balcony, pool, gym*
2	*balcony, gym*
3	*balcony, pool*

kindergarten	description
A	*German, indoor-playground*
B	*German, outdoor-playground*
C	*German, outdoor-playground*
D	*outdoor playground*

Figure 1: Motivation example

1 INTRODUCTION

The proliferation of geo-positioning technologies and services has resulted in an abundance of geospatial datasets augmented with text information. Examples include PoI datasets from online map and social websites (e.g., Yelp and FourSquare), geo-tagged photos with text description in photo sharing websites (e.g., Flickr and Instagram), geo-tagged micro-blogs (e.g., Twitter) and news. In this context, previous work for spatial keyword search has mainly focused on three types of queries [4, 7, 9]. First, given a location ℓ and a set of keywords ψ, the *k-NN (Boolean) Containment* query in [6, 13, 24], returns the k spatio-textual objects closely located to ℓ, whose text description contains all keywords in ψ. Second, instead of ranking the objects solely on their spatial distance to the query location ℓ, the *ranking* or *top-k* query in [11, 17, 21, 23], retrieves the k objects with the highest spatio-textual relevance, defined as a combination of the objects' distance to ℓ and the similarity of their text description to set ψ. Last, the *(Boolean) Range* query studied in [8, 10, 14, 16, 22, 25], retrieves all objects within distance ϵ from location ℓ, whose text description contains all keywords in set ψ.

The above queries involve a *single* keyword search with respect to a given set of keywords ψ. The difference lies on how the objects are ranked/filtered, i.e., either considering exclusively their distance to the query location ℓ or by also measuring the relevance of their description to set ψ. In some cases however, users are interested in posing *multiple* keyword searches and then *spatially combine* their results. Consider for example the scenario in Figure 1. A German-speaking family is planning their relocation to Brooklyn, New York. They are interested in an apartment which (1) offers specific amenities, e.g., a *balcony* and a private *gym*, and (2) is conveniently located nearby a *German*-speaking kindergarten, e.g., less than $1km$ away, with an *outdoor-playground.* The family's search for an ideal apartment can be formulated as two independent keyword searches (potentially issued on separate data sources), using set ψ_a = {*balcony, gym*} for the apartments and ψ_k = {*German, outdoor playground*} for the kindergartens. Their results are then spatially joined with respect to the $1km$ distance constraint. Kindergartens B, C in Figure 1 offer a *German*-speaking program and an *outdoor-playground.* At the same time, apartments 1, 2 match the family's criteria for a *balcony* and a *gym*, but only apartment 2 will be recommended, as apartment 1 is located over $1km$ away from the qualifying kindergartens.

To our knowledge, the above scenario is not considered by existing works in spatial keyword search. It is also not covered by spatio-textual joins [2], where objects are joined on their distance in space and their text similarity, or by collective spatio-textual search [5], where groups of objects that collectively cover a *single* set of keywords are returned. Only, spatial rank joins [19, 20] relate to our work, but in this case ranked lists of objects are spatially joined while in our case the keyword search results are completely unranked. To fill this gap, we propose two novel retrieval tasks that spatially combine the results of independent keyword searches. In brief, given collections of spatio-textual objects R, S and sets of keywords ψ_R, ψ_S, a *Spatially Joined Keyword Searches* query (SpaJKS) returns all object pairs $(r, s) \in R \times S$, located within a given distance threshold ϵ, so that the text description of r (resp. s) satisfies the keyword search over R (resp. S) based on ψ_R (resp. ψ_S). Intuitively, SpaJKS combines two keyword searches with a spatial distance join. The second task termed *Spatially Ranked Keyword Searches* query (SpaRKS) replaces the spatial ϵ-distance join of SpaJKS with a k-closest pairs operation, retrieving the k most closely located pairs of objects (r, s) that qualify the keyword searches. Intuitively, unlike SpaJKS, SpaRKS allows us to control the number of returned objects, which otherwise can be overwhelming large, where in reality, only a small fraction is actually reviewed by the users [15].

The key contributions of this paper are summarized as follows:

- Section 2 formally introduces the problem of spatially combining keyword searches as a novel, interesting type of spatial keyword search.

- Given two collections of spatio-textual objects, we define two variations, termed the *Spatially Joined Keyword Searches* (SpaJKS) and the *Spatially Ranked Keyword Searches* (SpaRKS) queries.
- Section 3 discusses five evaluation strategies for SpaJKS and SpaRKS queries, considering different types of indexing structures and query processing techniques.
- Section 4 conducts an experimental analysis using both real-world and synthetic datasets to compare and analyse the efficiency of the presented evaluation strategies.

2 PRELIMINARIES

We define a spatio-textual object o as a $\langle o.id, o.loc, o.text\rangle$ triplet, modeling the identity, the location, and the text description of o, respectively. Entry $o.loc$ takes values from the 2-dimensional space, while $o.text$ is a set of terms from a finite global dictionary D. We next revisit three query operations, which act as the building blocks for the problem at hand.

Definition 2.1 (Keyword Search). Let O be a collection of spatio-textual objects. Given a set of keywords $\psi = \{k_1, \ldots, k_m\}$, a *keyword search* query (KSearch) returns every object in O whose text description contains all keywords from set ψ. Formally:

$$\mathsf{KSearch}(O, \psi) = \{o \in O : \psi \subseteq o.text\}$$

For every pair of objects o_1 and o_2, we denote their distance in the 2-dimensional space with respect to $o_1.loc$ and $o_2.loc$, as $dist(o_1, o_2)$. Without loss of generality, we consider the Euclidean distance of the objects for the rest of this paper.

Definition 2.2 (Spatial Distance Join). Let R and S be two collections of spatio-textual objects. Given a threshold ϵ, a *spatial distance join* query (SpJoin) returns all pairs of objects $(r, s) \in R \times S$ located within a distance that does not exceed ϵ. Formally:

$$\mathsf{SpJoin}(R, S, \epsilon) = \{(r, s) \in R \times S : dist(r, s) \leq \epsilon\}$$

Definition 2.3 (k-Closest Pairs). Let R, S be two collections of spatio-textual objects. Given a positive integer k, a *k-closest pairs* query (CPairs) returns the k pairs of objects $(r, s) \in R \times S$ with the lowest distances. Formally:

- $|\mathsf{CPairs}(R, S, k)| = k$, and
- $\mathsf{CPairs}(R, S, k) = \{(r, s) \in R{\times}S : \forall (r', s') \notin \mathsf{CPairs}(R, S, k), dist(r', s') \geq dist(r, s)\}$

We now define the two variations to the problem of *spatially combining keyword searches.*

Definition 2.4 (Spatially Joined Keyword Searches). Let R, S be two collections of spatio-textual objects. Given two sets of keywords ψ_R, ψ_S and a spatial distance threshold ϵ, a *Spatially Joined Keyword Searches* query (SpaJKS), returns all pairs of objects $(r, s) \in \mathsf{SpJoin}(R', S', \epsilon)$ with $R' = \mathsf{KSearch}(R, \psi_R)$ and $S' = \mathsf{KSearch}(S, \psi_S)$.

Definition 2.5 (Spatially Ranked Keyword Searches). Let R, S be two collections of spatio-textual objects. Given two sets of keywords ψ_R, ψ_S and a positive integer k, a *Spatial Ranked Keyword Searches* query (SpaRKS), returns the k pairs of objects $(r, s) \in \mathsf{CPairs}(R', S', k)$ with $R' = \mathsf{KSearch}(R, \psi_R)$ and $S' = \mathsf{KSearch}(S, \psi_S)$.

Without loss of generality, we introduced above both SpaJKS and SpaRKS as binary operations but they can be straightforwardly extended to involve multiple input collections and keyword searches. However, we leave the efficient evaluation of these non-binary operations as future work, considering for instance techniques from multi-way spatial joins [18].

Table 1: Evaluation strategies

strategy	indexing used	
text-first	text	inverted indices $\mathcal{I}_R$, $\mathcal{I}_S$
spatial-first	spatial	R-trees $\mathcal{T}_R$, $\mathcal{T}_S$
spatial probing	spatial & text	inverted index $\mathcal{I}_R$, R-tree $\mathcal{T}_S$
spatio-textual probing	text & spatio-textual	inverted index $\mathcal{I}_R$, IR-tree $\mathcal{H}_S$
join-based	spatio-textual	IR-trees $\mathcal{H}_R$, $\mathcal{H}_S$
		spatio-textual grid $G_{R,S}$

ALGORITHM 1: Text-first strategy for SpaJKS

```
Input  : inverted indices $\mathcal{I}_R$, $\mathcal{I}_S$, keyword sets $\psi_R$, $\psi_S$, threshold $\epsilon$
Output : all object pairs $(r, s) \in$ SpaJKS($R$, $S$, $\psi_R$, $\psi_S$, $\epsilon$)
1 $R' \leftarrow$ SetContainmentQuery($\mathcal{I}_R$, $\psi_R$); ▹ Keyword search on R [12]
2 $S' \leftarrow$ SetContainmentQuery($\mathcal{I}_S$, $\psi_S$); ▹ Keyword search on S [12]
3 sort $R'$ and $S'$;                                ▹ Sort in one dimension
4 output all $(r, s) \in$ PlaneSweepJoin($R'$, $S'$, $\epsilon$);  ▹ Compute SpJoin [1]
```

3 EVALUATING SpaJKS & SpaRKS QUERIES

We next study the efficient evaluation of SpaJKS and SpaRKS queries. We present five strategies depending on how the input collections are indexed. Sections 3.1 and 3.2 focus on SpaJKS while Section 3.3 discusses the necessary changes for SpaRKS. Table 1 summarizes all the different strategies in this section.

3.1 Using Spatial and/or Text Indexing

3.1.1 Text-first Strategy. We start off with a *text-first* strategy, assuming that both R, S collections are textually indexed. Under this setting, the SpaJKS query is evaluated in two phases. First, we use text indexing to evaluate the keyword searches of the query, i.e., to compute $R' = \mathsf{KSearch}(R, \psi_R)$ and $S' = \mathsf{KSearch}(S, \psi_S)$. Next, we spatially join the intermediate results in R', S' with respect to the distance threshold ϵ. Algorithm 1 is a high-level pseudocode of this text-first strategy. We assume that R, S are indexed by inverted indices $\mathcal{I}_R$, $\mathcal{I}_S$, respectively. Keyword searches $R' = \mathsf{KSearch}(R, \psi_R)$ and $S' = \mathsf{KSearch}(S, \psi_S)$ are evaluated with standard set containment techniques, e.g., the cross-cutting approach from [12]. Last, as R', S' are not spatially indexed[1], $\mathsf{SpJoin}(R', S', \epsilon)$ is efficiently computed by sorting R', S' in one dimension of the space and then applying a plane-sweep based join [1].

3.1.2 Spatial-first Strategy. If both collections are spatially indexed, we can alternatively prioritize the spatial predicate in SpaJKS. Under this, the *spatial-first* strategy first computes $\mathsf{SpJoin}(R, S, \epsilon)$. Then, the text descriptions for every reported pair (r, s) are compared against sets ψ_R, ψ_S, respectively, to determine whether both r, s objects qualify the corresponding keyword searches. To avoid redundant computations, the result of each $\psi_R \subseteq r.text$ and $\psi_S \subseteq s.text$ check is cached and thus, conducted only once. Algorithm 2 is a high-level pseudocode of the spatial-first strategy. The inputs are indexed by the $\mathcal{T}_R$, $\mathcal{T}_S$ R-trees and the $\mathsf{SpJoin}(R, S, \epsilon)$ ϵ-distance join is computed by synchronously traversing the trees as in [3].

3.1.3 Spatial-probing Strategy. For the third strategy, we assume that one of the input collections (e.g., R) is textually indexed and the other spatially (e.g., S). Under this setting, the spatial join predicate in SpaJKS can be evaluated via a set of spatial range

[1]Note that even if the input R, S collections were also spatially indexed, R', S' would not be because they stem from the preceding text searches.

ALGORITHM 2: Spatial-first strategy for SpaJKS

```
Input   : R-trees $\mathcal{T}_R$, $\mathcal{T}_S$, keyword sets $\psi_R$, $\psi_S$, threshold $\epsilon$
Output  : all object pairs $(r, s) \in$ SpaJKS($R$, $S$, $\psi_R$, $\psi_S$, $\epsilon$)
1 foreach $(r, s) \in$ RtreeJoin($\mathcal{T}_R$, $\mathcal{T}_S$, $\epsilon$) do      ▷ Compute SpJoin [3]
2     if $\psi_R \subseteq r.text$ and $\psi_S \subseteq s.text$ then      ▷ Keyword searches
3         output $(r, s)$;                                                  ▷ Update result
```

ALGORITHM 3: Spatial-probing strategy for SpaJKS

```
Input   : inverted index $\mathcal{I}_R$, R-tree $\mathcal{T}_S$, keyword sets $\psi_R$, $\psi_S$,
          threshold $\epsilon$
Output  : all object pairs $(r, s) \in$ SpaJKS($R$, $S$, $\psi_R$, $\psi_S$, $\epsilon$)
1 $R' \leftarrow$ SetContainmentQuery($\mathcal{I}_R$, $\psi_R$); ▷ Keyword search on $R$ [12]
2 foreach $r \in R'$ do
3     foreach $s \in$ DiskRangeQuery($\mathcal{T}_S$, $r.loc$, $\epsilon$) do      ▷ Range query
        on $S$
4         if $\psi_S \subseteq s.text$ then                                  ▷ Keyword search on $S$
5             output $(r, s)$;                                               ▷ Update result
```

queries. Specifically, we first compute $R' =$ KSearch(R, ψ_R) using the text indexing on R. Then, for each object $r \in R'$, we employ the spatial indexing on S to execute a disk-based range query of radius ϵ, centered at $r.loc$. To pair and output r with an s object returned by this range query we also need to check if $\psi_S \subseteq s.text$ holds; as before, the result of this test is cached. Algorithm 3 illustrates a high-level code of this *spatial-probing* strategy; the disk-based range queries are evaluated by traversing the $\mathcal{T}_S$ R-tree in a depth-first fashion.

3.2 Using Spatio-textual Indexing

Despite their simplicity, the above strategies suffer from the following shortcomings. The text- and spatial-first use one type of indexing and they are thus sensitive to the selectivity of the keyword searches and the spatial join predicate in SpaJKS, respectively. On one hand, for the text-first strategy, keyword searches of low selectivity will incur large intermediate results, and thus, an expensive follow-up spatial ϵ-distance join. On the other hand, the spatial-first strategy is negatively affected when SpJoin returns a large number of (r, s) pairs, not only due to the high processing cost of the join but also because of the large number of text checks required to output the final results. Last, the spatial-probing strategy does use both spatial and text indexing, but we expect to perform well when at least one the keyword searches is very selective, the results of which will be used to probe the spatial index on the other collection. Otherwise, the cost of computing SpJoin via a set of disk-based range queries will be prohibitively high.

In view of these shortcomings, we next explain how SpaJKS queries can benefit from *spatio-textual* indexing. By prioritizing neither of the query predicates, these strategies are able to prune objects both spatially and textually at the same time.

3.2.1 *Spatio-textual-probing Strategy.*

We first reconsider the probing strategy from Section 3.1.3 and replace the spatial indexing on S with spatio-textual. This *spatio-textual-probing* strategy executes a *text-aware* disk-based range query for each $r \in$ KSearch(R, ψ_R), to determine, at the same time, all objects $s \in S$ within ϵ distance from $r.loc$ with $\psi_S \subseteq s.text$. For this purpose, the S input collection is indexed by an IR-tree [11] which augments every node of a traditional R-tree with an inverted index. Intuitively, while traversing the tree, we only visit nodes whose MBR overlaps with the current disk-based range and their text

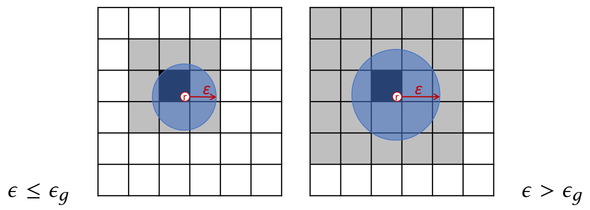


Figure 2: Spatially joinable cells; current cell c highlighted in black, cell edge length denoted by ϵ_g

description contains all keywords in ψ_S; the text description of a node is the union of the text descriptions for all objects indexed under its subtree. We efficiently identify such nodes using the incorporated inverted indices.

3.2.2 *Join-based Strategy.*

Last, we consider spatio-textual indexing under a *join-based* evaluation strategy for SpaJKS queries. We devise two versions of this strategy employing different spatio-textual indexing. For the first, we assume that the objects are indexed by two IR-trees. Under this, we extend the R-tree join algorithm of [3] to incorporate the keyword searches on the ψ_R, ψ_S sets. Intuitively, we consider only pairs of nodes (n_R, n_S) whose MBRs overlap and their text descriptions qualify the keyword searches; for the latter, we use again the inverted indices in the IR-tree nodes.

For the second version, we replace the IR-trees with the spatio-textual grid partitioning from [2].[2] In brief, the 2-dimensional space is divided by a uniform $m \times m$ grid; every object is assigned to exactly one cell of this grid. The objects of a grid cell c are then textually indexed by local inverted indices $\mathcal{I}^c_R$, $\mathcal{I}^c_S$. Given a SpaJKS$(R, S, \psi_R, \psi_S, \epsilon)$ query, we first determine which combinations of grid cells contain spatially joinable objects; we distinguish between two cases. If the query threshold ϵ is equal or smaller than the edge length of the grid cells, denoted by ϵ_g, then every r object inside a cell c can be spatially joined only with s objects from c itself and the 8 cells adjacent to c; the rest of the grid can be ignored.[3] Otherwise, if $\epsilon > \epsilon_g$, we extend this joinable area to include the next ring(s) of neighboring cells until the edge length of this area exceeds ϵ. Figure 2 illustrates the two cases. Under this, to evaluate the SpaJKS query, it suffices to traverse the spatio-textual grid and compute for each cell c, a set of SpaJKS$(R_c, S_j, \psi_R, \psi_S, \epsilon)$ *mini*-queries, where partition R_c stores all r objects assigned to c and S_j denotes the S partitions of the joinable cells to c. These mini-queries are evaluated using the text-first strategy. As a cell partition, e.g., S_j, is involved into multiple mini-queries, we avoid redundant computations by maintaining a sorted copy of $S'_j =$ KSearch(S_j, ψ_S).

3.3 The Case of SpaRKS Queries

Finally, we discuss how our strategies are extended for SpaRKS. First, we use a top-k list L to maintain the best object pairs found so far; initially, L is empty. Until the first k object pairs (r, s) that qualify the keyword searches are determined, all strategies are unable to perform any spatial pruning. After this point, we use the highest distance in L as threshold ϵ to perform spatial pruning similar to the SpaJKS case. Contrary though to SpaJKS, this threshold decreases as new qualifying pairs with lower distances are identified. Second, we prioritise the query evaluation

[2]In [2], the grid is built online as the extent of every cell equals the query threshold ϵ. For SpaJKS queries, we assume the grid is of fixed granularity and constructed offline or already exists.
[3]Special cases arise for the cells located at the corners or the borders of the space, where fewer than 9 cells are considered

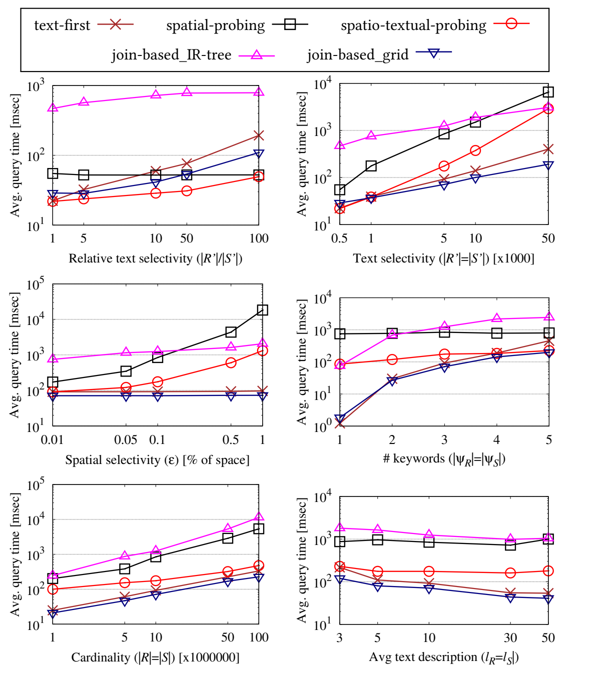


Figure 3: Synthetics: defaults, $|R'|/|S'| = 1$, $|R'| = |S'| = 5k$, $\epsilon = 0.1\%$, $|\psi_R| = |\psi_S| = 3$, $|R| = |S| = 10m$, $l_R = l_S = 10$

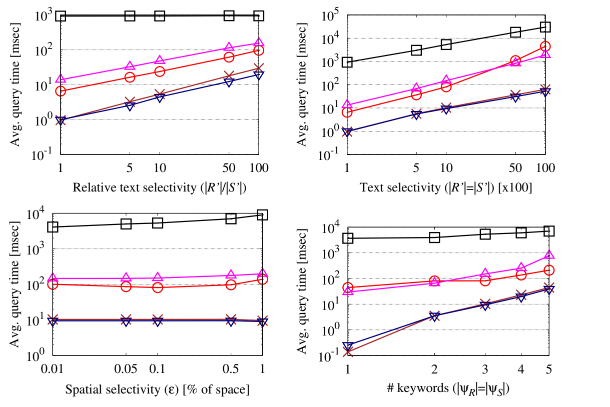


Figure 4: Polybot: defaults, $|R'|/|S'| = 1$, $|R'| = |S'| = 1000$, $\epsilon = 0.1\%$, $|\psi_R| = |\psi_S| = 3$

according to *dist*, which allows us to early terminate the process without computing the entire $R \times S$ product. Strategies that employ R-trees or IR-trees traverse now the trees in a best-first fashion, examining nodes or pairs of nodes by distance in increasing order. Last, the text-first and the join-based strategy with the spatio-textual grid rely on the sorting to prioritize the search.

4 EXPERIMENTAL ANALYSIS

Our evaluation was conducted on a dual Intel(R) Xeon(R) Gold 6130 CPU clocked at 2.10GHz with 750GBs of RAM. All strategies were implemented in C++, compiled using gcc (v4.11.2) with -O3.

Datasets. We experimented with both real and synthetic spatio-textual objects; note that all data (including the indices) resided in main memory. Specifically, Polybot real dataset [10] contains 6.1M crawled Web pages whose locations were assigned via geo-coding; the text description of every page contains 200 terms on average. We generated the synthetic datasets varying their cardinality from 1m to 100m objects, and the average length of the text description per object from 3 to 50. The locations of the objects follow a clustered distribution in the $[0, 1]^2$ space, with 10 randomly selected locations as cluster centers, while the frequencies of the terms follow a zipfian distribution over a 100k global dictionary.

Tests. For each evaluation strategy, we measured its response time over $10k$ queries, while varying their spatial selectivity via threshold ϵ (as a percentage of the space), the selectivity of the keyword searches via the cardinality of the intermediate results $|R'| = |\text{KSearch}(R, \psi_R)|$, $|S'| = |\text{KSearch}(S, \psi_S)|$, and the number of keywords $|\psi_R|$, $|\psi_S|$. To better understand the effect of the text selectivity, we also vary the relative selectivity of the keyword searches in $|R'|/|S'|$ ratio. In every test, we vary one paramater and fix the rests in their default. Without loss of generality, we use the same collection as R, S inputs ($R \equiv S$). Last, we assume that the necessary indices for each strategy pre-exist (for other types of spatial, text or spatio-textual queries), i.e., R-trees/IR-trees of a 4KBs page and a 50x50 spatio-textual grid for *join-based_grid*, and thus, we focus only on query response times.

Results. Due to lack of space, we provide plots only for SpaJKS queries; the results for SpaRKS follow similar trends. Figure 3 and 4 report the response times of the strategies for Synthetic and the Polybot collections. Also, we omit *spatial-first* as its times were orders of magnitude higher than the rest. Essentially, we can classify the strategies into 3 tiers based on their performance. At the bottom tier (worst performance), we always find *spatial-first* and depending on the SpJoin cost either *join-based_IR-tree* or the *spatial-probing*. *Spatial-textual-probing* is in general competitive and so placed in the second tier, especially on synthetics and when varying the$|R'|/|S'|$ ratio and one of the keyword searches is very selective. Compared to *spatial-probing*, *spatio-textual-probing* is faster in the majority of the tests. Recall that spatio-textual indexing allows us to perform both spatial and textual pruning while probing the tree. The top tier contains *text-first* and *join-based_grid* which in almost of all tests outperform the rest; exceptions arise in extreme cases, e.g., in Synthetics and $|R'|/|S'| = 100$ where the big difference in selectivity of the keyword searches benefits the probing approaches. Although *text-first* benefits from the nature of SpaRKS, i.e., executing the keyword searches first may significantly reduce the number of candidates, *join-based_grid* is in fact the fastest strategy overall. This is because *join-based_grid* accelerates the keyword searches, sorting and join by splitting them into small tasks (mini-queries).

5 CONCLUSIONS

We introduced the tasks of spatially joined and spatially ranked keyword searches as novel instances of spatial keyword search. For their efficient evaluation, we discussed multiple strategies relying on spatial, text or spatio-textual indexing. Our tests on both real and synthetic datasets showed that the join-based approach using a spatio-textual grid inspired by [2] is consistently the fastest. For the future, we plan to investigate the case of spatially combining multiple keyword searches and extend our work to consider other types of text search, e.g., similarity search.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the computing time granted on the supercomputer Mogon at Johannes Gutenberg University Mainz (hpc.uni-mainz.de).

REFERENCES

[1] Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. 1998. Scalable Sweeping-Based Spatial Join. In *VLDB*. 570–581.

[2] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. 2012. Spatio-textual similarity joins. *Proc. VLDB Endow.* 6, 1 (2012), 1–12.

[3] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient Processing of Spatial Joins Using R-Trees. In *ACM SIGMOD*. 237–246.

[4] Xin Cao, Lisi Chen, Gao Cong, Christian S. Jensen, Qiang Qu, Anders Skovsgaard, Dingming Wu, and Man Lung Yiu. 2012. Spatial Keyword Querying. In *ER*. 16–29.

[5] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *SIGMOD*. 373–384.

[6] Ariel Cary, Ouri Wolfson, and Naphtali Rishe. 2010. Efficient and Scalable Method for Processing Top-k Spatial Boolean Queries. In *SSDBM*. 87–95.

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

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

[9] Zhida Chen, Lisi Chen, Gao Cong, and Christian S. Jensen. 2021. Location- and keyword-based querying of geo-textual data: a survey. *VLDB J.* 30, 4 (2021), 603–640.

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

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

[12] Dong Deng, Chengcheng Yang, Shuo Shang, Fan Zhu, Li Liu, and Ling Shao. 2019. LCJoin: Set Containment Join via List Crosscutting. In *IEEE ICDE*. 362–373.

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

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

[15] Thorsten Joachims, Laura A. Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. *ACM Trans. Inf. Syst.* 25, 2 (2007), 7.

[16] Ali Khodaei, Cyrus Shahabi, and Chen Li. 2010. Hybrid Indexing and Seamless Ranking of Spatial and Textual Features of Web Documents. In *DEXA*. 450–466.

[17] 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.

[18] Nikos Mamoulis and Dimitris Papadias. 2001. Multiway spatial joins. *ACM Trans. Database Syst.* 26, 4 (2001), 424–475.

[19] Shuyao Qi, Panagiotis Bouros, and Nikos Mamoulis. 2013. Efficient Top-k Spatial Distance Joins. In *SSTD*. 1–18.

[20] Shuyao Qi, Panagiotis Bouros, and Nikos Mamoulis. 2020. Top-k spatial distance joins. *GeoInformatica* 24, 3 (2020), 591–631.

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

[22] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. 2005. Spatio-textual Indexing for Geographical Search on the Web. In *SSTD*. 218–235.

[23] Dingming Wu, Gao Cong, and Christian S. Jensen. 2012. A framework for efficient spatial web object retrieval. *VLDB J.* 21, 6 (2012), 797–822.

[24] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2013. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*. 901–912.

[25] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. 2005. Hybrid index structures for location-based web search. In *CIKM*. 155–162.