# On Processing Top-k Spatio-Textual Preference Queries

George Tsatsanifos
Knowledge and Database Systems Laboratory
National Technical University of Athens
gtsat@dblab.ntua.ece.gr

Akrivi Vlachou
Institute for the Management of Information Systems R.C. "Athena"
avlachou@imis.athena-innovation.gr

## ABSTRACT

In this paper we propose a novel query type, termed *top-k spatio-textual preference query*, that retrieves a set of spatio-textual objects ranked by the goodness of the facilities in their neighborhood. Consider for example, a tourist that looks for "hotels that have nearby a highly rated Italian restaurant that serves pizza". The proposed query type takes into account not only the spatial location and textual description of spatio-textual objects (such as hotels and restaurants), but also additional information such as ratings that describe their quality. Moreover, spatio-textual objects (i.e., hotels) are ranked based on the features of facilities (i.e., restaurants) in their neighborhood. Computing the score of each data object based on the facilities in its neighborhood is costly. To address this limitation, we propose an appropriate indexing technique and develop an efficient algorithm for processing our novel query. Moreover, we extend our algorithm for processing spatio-textual preference queries based on alternative score definitions under a unified framework. Last but not least, we conduct extensive experiments for evaluating the performance of our methods.

## 1. INTRODUCTION

An increasing number of applications support location-based queries, which retrieve the most interesting spatial objects based on their geographic location. Recently, spatio-textual queries have lavished much attention, as such queries combine location-based retrieval with textual information that describes the spatial objects. Most of the existing queries only focus on retrieving objects that satisfy a spatial constraint ranked by their spatio-textual similarity to the query point. However, in addition users are quite often interested in spatial objects (*data objects*) based on the quality of other facilities (*feature objects*) that are located in their vicinity. Feature objects are typically described by *non-spatial* attributes such as quality or rating, in addition to the *textual description*. In this paper, we propose a novel and more expressive query type than existing spatio-textual queries, called *top-k spatio-textual preference query*, for ranked retrieval of data objects based the textual relevance and the non-spatial score of feature objects in their neighborhood.

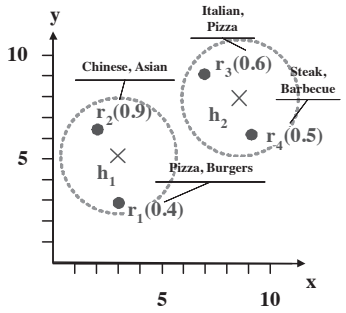



**Figure 1: Motivating example.**

Consider for example, a tourist that looks for *"hotels that have nearby a highly rated* ***Italian*** *restaurant that serves* ***pizza****"*. Figure 1 depicts a spatial area containing hotels (data objects) and restaurants (feature objects). The quality of the restaurants based on existing reviews is depicted next to the restaurant. Each restaurant also has textual information in the form of keywords extracted from its menu, such as pizza or steak, which describes additional characteristics of the restaurant. The tourist also specifies a spatial constraint (in the figure depicted as a range around each hotel) to restrict the distance of the restaurant to the hotel. Obviously, the hotel $h_2$ is the best option for a tourist that poses the aforementioned query. In the general case, more than one type of feature objects may exist in order to support queries such as *"hotels that have nearby a good* ***Italian*** *restaurant that serves* ***pizza*** *and a cheap coffeehouse that serves* ***muffins****"*. Even though spatial preference queries have been studied before [16, 17, 14], their definition ignores the available textual information. In our example, the spatial preference query would correspond to a tourist that searches for *"hotels that are nearby a good restaurant"* and the hotel $h_1$ would always be retrieved, irrespective of the textual information.

In this paper, we define top-$k$ spatio-textual preference queries and provide efficient algorithms for processing this novel query type. A main challenge compared to traditional spatial preference queries [16, 17, 14] is that the score of a data object changes depending on the query keywords, which renders techniques that rely on materialization (such





10.5441/002/edbt.2015.38

as [14]) not applicable. Most importantly, processing spatial preference queries is costly in terms of both I/O and execution time [16, 17]. Thus, extending spatial preference queries for supporting also textual information is challenging, since the new query type is more demanding due to the additional textual descriptions.

A straightforward algorithm for processing spatio-textual preference queries is to compute the *spatio-textual preference score* for each data object and then report the $k$ data objects with the highest score. We call this approach *Spatio-Textual Data Scan (STDS)* and examine it as a baseline, while our main focus is to reduce the cost required for computing the spatio-textual score of a data object.

Moreover, we develop an efficient and scalable algorithm, called *Spatio-Textual Preference Search (STPS)*, for processing spatio-textual preference queries. *STPS* follows a different strategy than *STDS*, as it retrieves highly ranked feature objects first, and then searches data objects in their spatial neighborhood. Intuitively, data objects located in the neighborhood of highly ranked feature objects are good candidates for inclusion in the top-$k$ result set. The main challenge tackled with *STPS* is determining efficiently the best feature objects from all feature sets that do not violate the spatial constraint.

To further improve the performance of our algorithms, we develop an appropriate indexing technique called *SRT-index*, that not only indexes the spatial location, the textual description and the non-spatial score, but in addition takes them equally into consideration during the index creation. Finally, we extend our algorithm for processing spatio-textual preference queries based on alternative score definitions under a unified framework. To summarize the contribution of this paper are:

- We propose a novel query type, called top-$k$ spatio-textual preference query, that ranks the data objects based on the quality and textual relevance of facilities (*feature objects*) located in their vicinity.
- A novel indexing technique called *SRT-index* is presented that is beneficial for processing spatio-textual preference queries.
- We present two algorithms for processing spatio-textual preference queries, namely *Spatio-Textual Data Scan (STDS)* and *Spatio-Textual Preference Search (STPS)*.
- We extend our algorithm *STPS* for processing spatio-textual preference queries based on alternative score definitions under a unified framework.
- We conduct an extensive experiment evaluation for studying the performance of our proposed algorithms and indexing technique.

The rest of this paper is organized as follows: Section 2 overviews the relevant literature. In Section 3, we define the spatio-textual preference query. Our novel indexing technique (*SRT-index*) is presented in Section 4. In Section 5 we describe our baseline algorithm, called spatio-textual data scan (*STDS*). An efficient algorithm, called Spatio-Textual Preference Search (*STPS*), is proposed in Section 6. Moreover, we extend our algorithms for processing spatio-textual preference queries based on alternative scores in Section 7. We present the experimental evaluation in Section 8 and we conclude in Section 9.

## 2. RELATED WORK

Recently several approaches have been proposed for spatial-keyword search. In [8], the problem of distance-first top-$k$ spatial keyword search is studied. To this end, the authors propose an indexing structure ($IR^2$-Tree) that is a combination of an R-Tree and signature files. The $IR$-Tree was proposed in another conspicuous work [6, 11], which is a spatio-textual indexing approach that employs a hybrid index that augments the nodes of an R-Tree with inverted indices. The inverted index at each node refers to a pseudo-document that represents all the objects under the node. During query processing, the index is exploited to retrieve the top-$k$ data objects, defined as the $k$ objects that have the highest spatio-textual similarity to a given data location and a set of keywords. Moreover, in [13] the *Spatial Inverted Index (S2I)* was proposed for processing top-$k$ spatial keyword queries. The S2I index maps each keyword to a distinct aggregated R-Tree or to a block file that stores the objects with the given term. All these approaches focus on ranking the data objects based on their spatio-textual similarity to a query point and some keywords. This is different from our work, which ranks the data objects based on textual relevance and a non-spatial score (quality) of the facilities in their spatial neighborhood. [5] provides an all-around evaluation of spatio-textual indices and reports on the findings obtained when applying a benchmark to the indices.

Spatio-textual similarity joins were studied in [1]. Given two data sets, the query retrieves all pairs of objects that have spatial distance smaller than a given value and at the same time a textual similarity that is larger than a given value. This differs from the top-$k$ spatio-textual preferences query, because the spatio-textual similarity join does not rank the data objects and some data objects may appear more than once in the result set. Prestige-based spatio-textual retrieval was studied in [2]. The proposed query takes into account both location proximity and prestige-based text relevance.

The $m$-closest keywords query [18] aims to find the spatially closest data objects that match with the query keywords. The authors in [3] study the spatial group keyword query that retrieves a group of data objects such that all query keywords appear in at least one data object textual description and such that objects are nearest to the query location and have the lowest inter-object distances. These approaches focus on finding a set of data objects that are close to each other and relevant to a given query, whereas in this paper we rank the data objects based on the facilities in their spatial neighborhood. In [4], the length-constrained maximum-sum region (LCMSR) query is proposed that returns a spatial-network region of constrained size that is located within a general region of interest and that best matches query keywords.

Ranking of data objects based on their spatial neighborhood without supporting keywords has been studied in [15, 7, 16, 17, 14]. Xia *et al.* studied the problem of retrieving the top-$k$ most influential spatial objects [15], where the score of a data object $p$ is defined as the sum of the scores of all feature objects that have $p$ as their nearest neighbor. Yang *et al.* studied the problem of finding an optimal location [7], which does not use candidate data objects but instead searches the space. Yiu *et al.* first considered computing the score of a data object $p$ based on feature objects in its spatial neighborhood from multiple feature sets [16, 17]

and defined top-$k$ spatial preference queries. In another line of work, a materialization technique for top-$k$ spatial preference queries was proposed in [14] which leads to significant savings in both computational and I/O cost during query processing. The main difference is that our novel query is defined in addition by a set of keywords that express desirable characteristics of the feature objects (like "pizza" for a feature object that represents a restaurant).

# 3. PROBLEM STATEMENT

Given an *object* dataset $O$ and a set of $c$ *feature* datasets $\{F_i \mid i \in [1, c]\}$, in this paper, we address the problem of finding $k$ data objects that have in their spatial proximity highly ranked feature objects that are relevant to the given query keywords. Each data object $p \in O$ has a spatial location. Similarly, each feature object $t \in F_i$ is associated with a spatial location but also with a *non-spatial score* $t.s$ that indicates the goodness (quality) of $t$ and its domain of values is the range $[0, 1]$. Moreover, $t$ is described by set of keywords $t.\mathcal{W}$ that capture the textual description of the feature object $t$. Figure 2 depicts an example of a set of feature objects that represent restaurants and shows the non-spatial score and the textual description. Table 1 provides an overview of the symbols used in this paper.

| Symbol | Description |
|---|---|
| $O$ | Set of data objects |
| $p$ | Data object, $p \in O$ |
| $c$ | Number of feature sets |
| $F_i$ | Feature sets, $i \in [1, c]$ |
| $t$ | Feature object, $t \in F_i$ |
| $t.s$ | Non-spatial score of $t$ |
| $t.\mathcal{W}$ | Set of keywords of $t$ |
| $dist(p, t)$ | Distance between $p$ and $t$ |
| $sim(t, \mathcal{W})$ | Textual similarity between $t$ and $\mathcal{W}$ |
| $s(t)$ | Preference score of $t$ |
| $\tau_i(p)$ | Preference score of $p$ based on $F_i$ |
| $\tau(p)$ | Spatio-textual preference score of $p$ |

**Table 1: Overview of symbols.**

The goal is to find data objects that have in their vicinity feature objects that (i) are of high quality and (ii) are relevant to the query keywords posed by the user. Thus, the score of the feature object $t$ captures not only the non-spatial score of the feature, but its textual similarity to a user specified set of query keywords.

DEFINITION 1. *The* ***preference score*** $s(t)$ ***of feature object*** $t$ *based on a user-specified set of keywords* $\mathcal{W}$ *is defined as* $s(t) = (1 - \lambda) \cdot t.s + \lambda \cdot sim(t, \mathcal{W})$*, where* $\lambda \in [0, 1]$ *and* $sim()$ *is a textual similarity function.*

The textual similarity between the keywords of the feature and the set $\mathcal{W}$ is measured by $sim(t, \mathcal{W})$ and its domain of values is the range $[0, 1]$. The parameter $\lambda$ is the smoothing parameter that determines how much the score of the feature objects should be influenced by the textual information. For the rest of the paper, we assume that the textual similarity is equal to the Jaccard similarity between the keywords of the feature objects and the user-specified keywords: $sim(t, \mathcal{W}) = \frac{|t.\mathcal{W} \bigcap \mathcal{W}|}{|t.\mathcal{W} \bigcup \mathcal{W}|}$.

For example, consider the restaurants depicted in Figure 2. Given a set of keywords $\mathcal{W} = \{italian,\ pizza\}$ and $\lambda = 0.5$ the restaurant with the highest preference score is *Ontario's Pizza* with a preference score $s(r_6) = 0.9$, while the score of *Beijing Restaurant* is $s(r_1) = 0.3$, since none of the given keywords are included in the description of *Beijing Restaurant*.

Given a spatio-textual preference query $Q$ defined by an integer $k$, a range $r$ and $c$-sets of keywords $\mathcal{W}_i$, the preference score of a data object $p \in O$ based on a feature set $F_i$ is defined by the scores of feature objects $t \in F_i$ in its spatial neighborhood, whereas the overall spatio-textual score of $p$ is defined by taking into account all feature sets $F_i$, $1 \leq i \leq c$.

DEFINITION 2. *The* ***preference score*** $\tau_i(p)$ ***of data object*** $p$ *based on the feature set* $F_i$ *is defined as:* $\tau_i(p) = max\{s(t) \mid t \in F_i : dist(p, t) \leq r \text{ and } sim(t, \mathcal{W}_i) > 0\}$.

The $dist(p, t)$ denotes the spatial distance between data object $p$ and feature object $t$ and we employ the Euclidean distance function. Continuing the previous example, Figure 4 shows the spatial location of the restaurants in Figure 2 and a data point $p$ that represents a hotel. The preference score of $p$ based on the restaurants in its neighborhood (assuming $r = 3.5$ and $\mathcal{W} = \{italian,\ pizza\}$) is equal to the score of $r_6$ ($\tau_i(p) = s(r_6) = 0.9$), which is the best restaurant in the neighborhood of $p$.

DEFINITION 3. *The overall* ***spatio-textual preference score*** $\tau(p)$ ***of data object*** $p$ *is defined as:* $\tau(p) = \sum_{i \in [1,c]} \tau_i(p)$.

Figure 3 shows a second set of feature objects that represents coffeehouses. For a tourist that looks for a good hotel that has nearby a good Italian restaurant that serves pizza and a good coffeehouse that serves espresso and muffins, the score of $p$ would be $\tau(p) = s(r_6) + s(c_5) = 0.9 + 0.78233 = 1.6833$.

PROBLEM 1. Top-$k$ Spatio-Textual Preference Queries (STPQ)*: Given a query $Q$, defined by an integer $k$, a radius $r$ and $c$-sets of keywords $\mathcal{W}_i$, find the $k$ data objects $p \in O$ with the highest spatio-textual score $\tau(p)$.*

# 4. INDEXING

The main difference of top-$k$ spatio-textual preference queries to traditional spatio-textual search is that the ranking of a data object does not depend only on spatial location and textual information, but also on the non-spatial score of the feature object. In particular, the preference score $s(t)$ of feature object $t$ is defined by its textual description and its non-spatial score, while the spatial location is used as a filter for computing the preference score $\tau_i(p)$ of data object $p$. Thus, efficient indexing of the textual description and the non-spatial score of feature objects is a significant factor for designing efficient algorithms for the STPQ query.

## 4.1 Index Characteristics

In this paper, we assume that the data objects $O$ are indexed by an R-Tree, denoted as *rtree*. However, for the feature objects, it is important that the non-spatial score and the textual description are indexed additionally. Each dataset $F_i$ can be indexed by any spatio-textual index that relies on a spatial hierarchical index (such as the R-Tree). However, each entry $e$ of the index must in addition maintain: (i) the maximum value of $t.s$ of any feature object $t$ in the sub-tree, denoted as $e.s$, and (ii) a summary ($e.\mathcal{W}$) of

| | name | rating | x | y | textual description |
|---|---|---|---|---|---|
| $r_1$ | Beijing Restaurant | 0.6 | 1 | 2 | Chinese, Asian |
| $r_2$ | Daphne's Restaurant | 0.5 | 4 | 1 | Greek, Mediterranean |
| $r_3$ | Espanol Restaurant | 0.8 | 5 | 8 | Italian, Spanish, European |
| $r_4$ | Golden Wok | 0.8 | 2 | 3 | Chinese, Buffet |
| $r_5$ | John's Pizza Plaza | 0.9 | 8 | 4 | Pizza, Sandwiches, Subs |
| $r_6$ | Ontario's Pizza | 0.8 | 7 | 6 | Pizza, Italian |
| $r_7$ | Oyster House | 0.8 | 6 | 10 | Seafood, Mediterranean |
| $r_8$ | Small Bistro | 1.0 | 3 | 7 | American, Coffee, Tea, Bistro |

**Figure 2: Feature objects (Restaurants)**

| | name | rating | x | y | textual description |
|---|---|---|---|---|---|
| $c_1$ | Bakery & Cafe | 0.6 | 4 | 1 | Cake, Bread, Pastries |
| $c_2$ | Coffee House | 0.5 | 4 | 7 | Cappuccino,Toast, Decaf |
| $c_3$ | Coffe Time | 0.8 | 3 | 10 | Cake, Toast, Donuts |
| $c_4$ | Cafe Ole | 0.6 | 6 | 2 | Cappuccino, Iced Coffee, Tea |
| $c_5$ | Royal Coffe Shop | 0.9 | 5 | 5 | Muffins, Croissants,Espresso |
| $c_6$ | Mocha Coffe House | 1.0 | 10 | 3 | Macchiato, Espresso, Decaf |
| $c_7$ | The Terrace | 0.7 | 6 | 9 | Muffins, Pastries, Espresso |
| $c_8$ | Espresso Bar | 0.4 | 7 | 6 | Croissants, Decaf, Tea |

**Figure 3: Feature objects (Coffeehouses)**

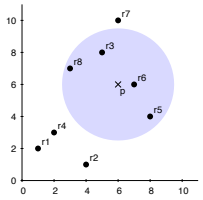



**Figure 4: An example of a $STPQ$ query.**

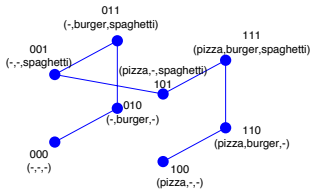



**Figure 5: Hilbert-based keyword ordering.**

all keywords of any feature $t$ in the sub-tree. To ensure correctness of our algorithms, there must exist an upper bound $\widehat{s}(e)$ such that for any $t$ stored in the sub-tree rooted by the entry $e$ it holds:

$$\widehat{s}(e) \geq s(t)$$

The above property guarantees that the preference score $s(t)$ of a feature object $t$ is bounded by the bound $\widehat{s}(e)$ of its ancestor node $e$. The efficiency of the algorithms directly depends on the tightness of this bound. In turn, this depends on the similarity between the textual description and the non-spatial score of the features objects that are indexed in the same node.

In the following, we propose an indexing technique that leads to tight bounds since objects with similar textual information and non-spatial score are stored in the same node of the index.

## 4.2 Indexing based on Hilbert Mapping

Our indexing approach maps the textual description of feature objects to a value based on the Hilbert curve. Let $w$ denote the number of distinct keywords in the vocabulary, then for each feature $t$ the keywords $t.\mathcal{W}$ can be represented as a binary vector of length $w$. For instance, assuming a vocabulary $\{pizza, burger, spaghetti\}$, we can use an active bit to declare the existence of the *"pizza"* keyword at the first place, *"burger"* at the second, and *"spaghetti"* at the last. Moreover, we suggest a mapping of the binary vector to a Hilbert value, denoted as $\mathcal{H}(t.\mathcal{W})$. For the above $w$=3 keywords, the defined order is 000,010,011,001,101,111,110 and 100. Figure 5 shows the ordering of the keywords based on the Hilbert values. The benefit of this order is that it ensures us that vectors with distance 1 have only one different keyword, while if the distance is $w'$, then the maximum number of different keywords is bound by $w'$. This means that consecutive vectors in the afore-described order have only few different keywords, which means that objects with sequential $\mathcal{H}$-values are highly similar also based on the Jaccard similarity function.

Using the Hilbert mapping of the textual information, each feature object $t$ can be represented as a point in the 4-dimensional space $\{t.x,\ t.y,\ t.s,\ \mathcal{H}(t.\mathcal{W})\}$. Our indexing technique, called *SRT-index*, uses a spatial index, such as a traditional R-Tree, that is built on the mapped 4-dimensional space. In terms of structure, the SRT-index resembles a traditional R-Tree that it is built on the spatial location, the non-spatial score (rating), and the Hilbert value of the keywords of the feature objects altogether. The only modification needed during the index construction is the method used for updating the Hilbert values of a node. When the Hilbert value of a node is updated because a new object is added, then the previous Hilbert value as well the Hilbert value of the new object are mapped to binary vectors, the disjunction of the binary vectors is computed, mapped to a new Hilbert value and stored in the node. Notably, the exact spatial index used for indexing the mapped space does not affect the correctness of our algorithms, but only their performance. In our experimental evaluation, we use bulk insertion [9] on our novel indexing technique.

During query processing the bound $\widehat{s}(e)$ of a node $e$ can be set as:

$$\widehat{s}(e) = (1-\lambda) \cdot e.s + \lambda \cdot \frac{|e.\mathcal{W} \bigcap \mathcal{W}|}{|\mathcal{W}|}$$

where $\mathcal{W}$ is the set of query keywords, while $e.\mathcal{W}$ is the set of all keywords of all feature objects $t$ indexed by the node $e$. The set $e.\mathcal{W}$ is computed based on the Hilbert mapping and the aggregated Hilbert value $\mathcal{H}(e.\mathcal{W})$ stored in the node entry $e$ of the SRT-tree. It holds that $\widehat{s}(e) \geq s(t)$.

To summarize, the SRT-index overcomes the difficulty that other indexing approaches face, being unable to identify in advance what are the branches of the index that store highly ranked and relevant feature objects to the query. The reason is that this indexing mechanism can identify effectively the promising parts of the hierarchical structure at a low cost, since during the index construction the similarity of the spatial location, the non-spatial score, as well as the textual description are taken into account.

# 5. SPATIO-TEXTUAL DATA SCAN (STDS)

Our baseline approach, called spatio-textual data scan ($STDS$), computes the spatio-textual score $\tau(p)$ of *each* data object $p \in O$ and then reports the $k$ data objects with the highest score. Algorithm 1 shows the pseudocode of $STDS$.

In more detail, for a data object $p$, its score $\tau_i(p)$ for every feature set $F_i$ is computed (lines 3-5). The details on this computation for range queries are described in Algorithm 2 that will be presented in the sequel. Interestingly, for some data objects $p$ we can avoid computing $\tau_i(p)$ for some feature sets. This is feasible because we can determine early that some data objects cannot be in the result set $R$. To achieve this goal, we define a threshold $\tau$ which is the $k$-th highest score of any data object processed so far. In addition, we define an upper bound $\widehat{\tau}(p)$ for the spatio-textual preference score $\tau(p)$ of $p$, which does not require knowledge of the preference scores $\tau_i(p)$ for all feature sets $F_i$: $\widehat{\tau}(p) = \sum_{i \in [1,c]} \begin{cases} \tau_i(p), & if\ \tau_i(p)\ is\ known \\ 1, & otherwise \end{cases}$. The algorithm tests the upper bound $\widehat{\tau}$ based on the already computed $\tau_i(p)$ against the current threshold (line 6). If $\widehat{\tau}$ is smaller than the current threshold, the remaining score computations are avoided. After computing the score of $p$, we test whether it belongs to $R$ (line 6). If this is case, the result set $R$ is updated (line 7), by adding $p$ to it and removing the data object with the lowest score (in case that $|R| > k$). Finally, if at least $k$ data objects have already been added to $R$, we update the threshold based on the $k$-th highest score (line 9).

**Algorithm 1:** *Spatio-Textual Data Scan (*STDS*)*

```
   Input: Query Q = (k, r, {W_i})
   Output: Result set R sorted based on τ(p)
1  R = ∅; τ = −1;
2  foreach p ∈ O do
3      for i = 1 ... c do
4          if τ̂(p) > τ then
5              τ_i(p) = F_i.computeScore(Q, p) ;
6      if τ(p) > τ then
7          update(R) ;
8          if |R| ≥ k then
9              τ = k^th score ;
10 return R ;
```

The remaining challenge is to compute efficiently the score based on the spatio-textual information of the feature objects. The goal is to reduce the number of disk accesses for retrieving feature objects that are necessary for computing the score of each element $p \in O$. Algorithm 2 shows the computation of preference score $\tau_i(p)$ for feature set $F_i$. First, the root entry is retrieved and inserted in a heap (line 1). The heap maintains the entries $e$ sorted based on their values $\widehat{s}(e)$. In each iteration (lines 2-11), the entry $e$ with the highest value $\widehat{s}(e)$ is processed, following a best-first approach. If $e$ is a data point and within distance $r$ from $p$ (line 5), then the score $\tau_i(p)$ of $p$ has been found and is returned (line 7). If $e$ is not a data point, then we expand it only if it satisfies the query constraints (line 9). More detailed, if the minimum distance of $e$ to $p$ is smaller or equal to $r$ and its textual similarity is larger than 0, $e$ is expanded and its child entries are added to the heap (line 11). Otherwise, the entire sub-tree rooted at $e$ can be safely pruned.

**Correctness and Efficiency:** Algorithm 2 always reports the correct score $\tau_i(p)$. The sorted access of the entries, combined with the property that the value $\widehat{s}(e)$ of the entry is an upper bound ensure its correctness. Moreover, it can be shown that Algorithm 2 expands the minimum number of entries, in the sense that if an entry that is expanded was not expanded, it could lead to computing a wrong score. This is because only entries with score higher than any processed feature object are expanded, and such entries may contain in their sub-tree a feature object with score equal to the score of the entry.

**Algorithm 2:** *Spatio-Textual Score Computation on $F_i$ (computeScore$(Q, p)$)*

```
   Input: Query Q, data object p
   Output: Score τ_i(p)
1  heap.push(F_i.root);
2  while (not heap.isEmpty()) do
3      e ← heap.pop() ;
4      if e is a data object then
5          if (dist(p, e) ≤ r) then
6              τ_i(p) = s(e) ;
7              return τ_i(p) ;
8      else
9          if (mindist(p, e) ≤ r) and (sim(e, W_i) > 0) then
10             for childEntry in e.childNodes do
11                 heap.push(childEntry) ;
```

**Performance improvements:** The performance of *STDS* can be improved by processing the score computations in a batch. Instead of a single data object $p$, a set of data objects $\mathcal{P}$ can be given as input to Algorithm 2. Then, an entry is expanded if the distance for *at least* one $p$ in $\mathcal{P}$ is smaller than $r$. When a feature object is retrieved, for any $p$ for which the distance is smaller than $r$ the score is computed and those data objects $p$ are removed from $\mathcal{P}$. The same procedure is followed until either the heap or $\mathcal{P}$ is empty. Algorithm 1 can be easily modified to invoke Algorithm 2 for all data objects in the same leaf entry of the R-tree (*rtree*) that indexes the data objects $O$. For sake of simplicity, we omit the implementation details, even though we use this improved modification in our experimental evaluation.

# 6. SPATIO-TEXTUAL PREFERENCE SEARCH (STPS)

In this section we propose a novel and efficient algorithm, called Spatio-Textual Preference Search (*STPS*), for processing spatio-textual preference queries. *STPS* follows a different strategy than *STDS*, as it involves two major steps, namely finding highly ranked feature objects first, and then, retrieving data objects in their spatial neighborhood. Intuitively, if we find a neighborhood in which highly ranked feature objects exist, then the neighboring data objects are naturally highly ranked as well.

## 6.1 Valid Combination of Feature Objects

In a nutshell, the goal is to find sets of feature objects $\mathcal{C} = \{t_1, t_2, \ldots, t_c\}$ where $t_i \in F_i$ $(1 \leq i \leq c)$, such that the spatio-textual preference score of each $t_i$ is as high as possible and the feature objects are located in nearby locations.

In the general case, a data object may be highly ranked even in the case where a certain kind of feature object does not exist in its neighborhood, though feature objects of other kinds might compensate for this. For example, consider the extreme case where all data objects have only one type of feature object in their spatial neighborhood. For ease of pre-

**Algorithm 3:** *Spatio-Textual Preference Search (STPS)*

**Input**: Query $Q$
**Output**: Result set $R$ sorted based on $\tau(p)$
1 **while** *($|R| \leq k$)* **do**
2 $\quad \mathcal{C} = nextCombination(Q)$ ;
3 $\quad R = R \cup$ getDataObjects($\mathcal{C}$) ;
4 **return** $R$ ;

sentation, we denote as $\emptyset$ a virtual feature object for which it holds that $dist(p, \emptyset) = 0$, $dist(t_i, \emptyset) = 0$ and $s(\emptyset) = 0$ $\forall t_i, p$. This virtual feature object is used for presenting unified definitions for the case where the spatio-textual score of the top-$k$ data objects is defined based on less than $c$ feature objects. More formally put, we define the concept of *valid combination* of feature objects as:

DEFINITION 4. *A valid combination of feature objects is a set $\mathcal{C} = \{t_1, t_2, \ldots, t_c\}$ such that (i) $\forall i$ $t_i \in F_i$ or $t_i = \emptyset$, and (ii) $dist(t_i, t_j) \leq 2r$ $\forall i, j$. The score of the valid combination $\mathcal{C}$ is defined as $s(\mathcal{C}) = \sum_{1 \leq i \leq c} s(t_i)$.*

The following lemma proves that it is sufficient to examine only the valid combinations $\mathcal{C}$ of feature objects in order to retrieve the result set of a top-$k$ spatio-textual preference query.

LEMMA 1. *The score of any data object $p \in O$ is defined by a valid combination of feature objects $\mathcal{C} = \{t_1, t_2, \ldots, t_c\}$, i.e., $\forall p : \exists \mathcal{C} = \{t_1, t_2, \ldots, t_c\}$ such that $\tau(p) = s(\mathcal{C})$*

PROOF. Let us assume that there exists $p$ such that: $\tau(p) = \sum_{i \in [1,c]} \tau_i(p)$ with $\tau_i(p) = \{s(t_i) \mid t_i \in F_i : dist(p, t_i) \leq r$ *and* $sim(t_i, \mathcal{W}_i) > 0\}$ and $\mathcal{C} = \{t_1, t_2, \ldots, t_c\}$ is not a valid combination of feature objects. Since $\mathcal{C} = \{t_1, t_2, \ldots, t_c\}$ is not a valid combination of feature objects, there exists $1 \leq i \neq j \leq c$ such that $dist(t_i, t_j) > 2r$ but also $dist(p, t_i) \leq r$ and $dist(p, t_j) \leq r$. Based on the triangular inequality it holds: $dist(t_i, t_j) \leq dist(p, t_i) + dist(p, t_j) \leq r + r \leq 2r$, which is a contradiction. □

## 6.2 STPS Overview

Algorithm 3 provides an insight to *STPS* algorithm. At each iteration, the following steps are followed: (i) a special iterator (line 2) returns successively the valid combinations of feature objects sorted based on their score (we discuss the details on the implementation of the iterator in the following subsection), (ii) up to $k$ data points in the spatial neighborhood of these features are retrieved (line 3). Data objects that have already been previously retrieved are discarded, while the remaining data objects $p$ have a score $\tau(p) = s(\mathcal{C})$ and can be returned to the user incrementally. If $k$ data objects have been returned to the user (line 1), the algorithm terminates without retrieving the remaining combinations of feature objects. Differently to the *STDS* algorithm, *STPS* retrieves only the data objects that most certainly belong to the result set.

## 6.3 Spatio-Textual Feature Objects Retrieval

Algorithm 4 shows the pseudocode for retrieving the valid combinations $\mathcal{C} = \{t_1, t_2, \ldots, t_c\}$ of feature objects sorted based on their spatio-textual preference score $s(\mathcal{C})$. We first give a scketch of our algorithm and then we will elaborate

**Algorithm 4:** *Spatio-Textual Feature Objects Retrieval (nextCombination(Q))*

**Input**: Query $Q$
$heap_i$: heap maintaining entries of $F_i$
$heap$: heap maintaining valid combinations of feature objects
$\mathcal{D}_i$: set of feature objects of $F_i$
**Output**: $\mathcal{C}$: valid combination with highest score
1 **while** *($\exists i$ :* ***not*** *$heap_i.isEmpty()$)* **do**
2 $\quad i \leftarrow$ nextFeatureSet() ;
3 $\quad e_i \leftarrow heap_i.pop()$ ;
4 $\quad$ **while** *(**not** $e_i$ is a data object)* **do**
5 $\quad\quad$ **for** *childEntry **in** $e_i.childNodes$* **do**
6 $\quad\quad\quad heap_i$.push(childEntry) ;
7 $\quad\quad e_i \leftarrow heap_i$.pop() ;
8 $\quad \mathcal{D}_i = \mathcal{D}_i \cup e_i$ ;
9 $\quad heap$.push(validCombinations($\mathcal{D}_1, \cdots, e_i, \cdots, \mathcal{D}_c$)) ;
10 $\quad min_i = s(e_i)$ ;
11 $\quad \tau = max_{1 \leq j \leq c}(max_1 + \cdots + min_j + \cdots + max_c)$ ;
12 $\quad \mathcal{C} \leftarrow heap$.top() ;
13 $\quad$ **if** *(score($\mathcal{C}$) $\geq \tau$)* **then**
14 $\quad\quad heap$.pop() ;
15 $\quad\quad$ **return** $\mathcal{C}$;

further on the details in the following of this section. In each iteration, a feature set $F_i$ is selected (line 2) based on a pulling strategy implemented by $nextFeatureSet()$. The spatio-textual index that stores the feature objects of the feature set $F_i$ is accessed and the feature objects $t_i$ are retrieved based on their score $s(t_i)$ that aggregates their non-spatial score, but also their textual similarity to the query keywords (lines 3-7). The retrieved feature objects are maintained in a list $\mathcal{D}_i$ (line 8) and are used to produce valid combinations $\mathcal{C}$ of feature objects (line 9). Moreover, a thresholding scheme is employed to decide when the combination with the highest score has been retrieved (lines 11-15).

We denote as $max_i$ the maximum score of $\mathcal{D}_i$ and $min_i$ the minimum score of $\mathcal{D}_i$. Thus, $min_i$ represents the best potential score of any feature object of $F_i$ that has not been processed yet. Moreover, in Algorithm 4 the variables $heap_i$, $\mathcal{D}_i$, $max_i$, $min_i$, and $heap$ are global variables. They are initialized as following $heap_i$: the root of $F_i$, $\mathcal{D}_i = \emptyset$ and $heap = \emptyset$, $min_i = \infty$. Variable $max_i$ is the score of the highest ranked feature object of $F_i$ and is set the first time the $F_i$ index is accessed.

**Accessing $F_i$:** In each iteration, Algorithm 4 accesses one spatio-textual index that stores the set $F_i$ (lines 3-7). The entries of the spatio-textual index responsible for the feature objects of $F_i$ are maintained in $heap_i$, which keeps the entries $e$ sorted based on $\widehat{s}(e)$. Moreover, for sake of simplicity, we assume that $heap_i.pop()$ will return a virtual feature object $t_i = \emptyset$ (with score equal to 0) as final object. In each iteration an entry $e_i$ of the spatio-textual index is retrieved from $heap_i$ (line 3). If the entry $e_i$ corresponds to a node of the index, the entry is expanded and its child nodes are added to the $heap_i$ (lines 5-6). Algorithm 4 continues retrieving from $heap_i$ entries, until an entry that is a feature object is retrieved (line 4). When an entry $e_i$ is retrieved that corresponds to a feature object, $e_i$ is inserted in the list $\mathcal{D}_i$ (line 8).

**Creation of $\mathcal{C}$:** After retrieving a new feature object $e_i$, new valid combinations $\mathcal{C}$ are created by combining $e_i$ with the previously retrieved feature objects $t_j$ maintained in the lists $\mathcal{D}_j$ (line 9). For this, the method *validCombinations* is called, which returns all combinations of the objects in $\mathcal{D}_j$ and $e_i$, by discarding combinations for which the condition $dist(t_i, t_j) \leq 2r \ \forall i, j$ does not hold. The new valid combinations are inserted in the *heap* (line 9) that maintains the valid combinations sorted based on their score $s(\mathcal{C})$.

**Thresholding scheme:** Algorithm 4 employs a thresholding scheme to determine if the current best valid combination can be returned as the valid combination with the highest score. The threshold $\tau$ represents the best score of any valid combination of feature objects that has not been examined yet. The best score of the next feature object $t_j$ retrieved from $F_j$ is equal to $min_j$, since the feature objects are accessed sorted based on $s(t_j)$. Obviously, for the remaining feature sets we assume that the new feature object $t_j$ is combined with the feature objects that have the highest score. Thus, $\tau = max_{1 \leq j \leq c}(max_1 + \cdots + min_j + \cdots + max_c)$ (line 11) is an upper bound of the score for any valid combination that has not been examined yet. In line 13, we test whether the best combination of feature objects in the *heap* has a score higher or equal to the threshold $\tau$. If so, the best combination in the heap is the next valid combination with the best score. Otherwise, additional feature objects from feature sets $F_i$ have to be retrieved until it holds that the top element of the *heap* achieves a score which is higher than $\tau$.

**Pulling strategy:** In the following, we proposed an advanced pulling strategy that prioritizes retrieval from feature sets that have higher potential to produce the next valid combination $\mathcal{C}$. A simple alternative would be to access the different feature sets in a round robin fashion.

The order in which the feature objects of different feature sets are retrieved is defined by a pulling strategy, i.e., $nextFeatureSet()$ returns an integer between 1 and $c$ and defines the pulling strategy. In addition, $nextFeatureSet()$ never returns $i$ if $heap_i$ is empty.

DEFINITION 5. *Given $c$ sets of feature objects $\mathcal{D}_i$, the prioritized pulling strategy returns $m$ as the next feature set such that $\tau = max_1 + \cdots + min_m + \cdots + max_c$.*

The main idea of the prioritized pulling strategy is that in each iteration the feature set $F_m$ that is responsible for the threshold value $\tau$ is accessed. It is obvious that the only way to reduce $\tau$ is to reduce the $min_m$, since retrieval from the remaining feature sets cannot reduce $\tau$. Thus, retrieving the next tuple from the feature set $F_m$ may reduce the threshold $\tau$ and may produce new valid combinations that have a score equal to the current threshold.

## 6.4 Retrieval of Qualified Data Objects

In the following, we study the reciprocal actions taken upon the formation of a highly ranked combination of feature objects.

In Algorithm 3 (line 3) *getObjects($\mathcal{C}$)* is invoked to retrieve from *rtree* all data objects in the neighborhood of the feature objects in $\mathcal{C}$. This method starts from the root of the *rtree* and processes its entries recursively. Entries $e$ for which $\exists i$ such that $t_i \in \mathcal{C}$ with $dist(e, t_i) > r$ are discarded. The remaining entries are expanded until all objects $p$ for which it holds that $dist(p, t_i) \leq r$ are retrieved.

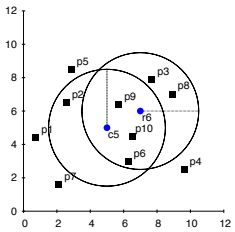



**Figure 6: Data objects within qualifying distance from $\mathcal{C} = \{r_6, c_5\}$.**

**Example.** *Consider for example the feature sets depicted in Figure 2 and in Figure 3. Given a query with $r = 3.5$, $\mathcal{W}_1 = \{italian,\ pizza\}$ and $\mathcal{W}_2 = \{espresso,\ muffins\}$, the restaurant and the coffeehouse with the highest scores are $r_6$ and $c_5$ respectively. Since it holds that $dist(r_6, c_5) \leq 2r$, the set $\mathcal{C} = \{r_6, c_5\}$ is a valid combination of feature objects. Assume that the set of data objects is $O = \{p_1, p_2, \ldots, p_{10}\}$ as depicted in Figure 6. For the data objects $p_6$, $p_9$ and $p_{10}$ it holds that $dist(p_i, c_5) \leq r$ and $dist(p_i, r_6) \leq r$, and their spatial-textual score is $\tau(p_6) = \tau(p_9) = \tau(p_{10}) = 1.6833$. These data objects are guaranteed to be the highest ranked data objects and can be immediately returned to the user. For $k \leq 3$, our algorithm terminates without examining other feature combinations.*

# 7. VARIANTS OF TOP-K SPATIO-TEXTUAL PREFERENCE QUERIES

In this section, we extend our algorithms for processing spatio-textual preference queries based on alternative score definitions under a unified framework. We provide formal definitions for the alternative score definitions, namely *influence preference score* and *nearest neighbor preference score*. Moreover, we discuss for all query types the necessary modifications to our query processing algorithms.

## 7.1 Influence-Based STPQ Queries

In contrast to the preference score defined in Definition 1 (in the following referred to as range score), in this section we define an alternative score that does not pose a hard constraint on the distance, but instead gradually reduces the score based on the distance. We call this variant *influence preference score*.

DEFINITION 6. *The **influence preference score** $\tau_i(p)$ of data object $p$ based on the feature set $F_i$ is defined as: $\tau_i(p) = max\{s(t) \cdot 2^{\frac{-dist(p,t)}{r}} \mid t \in F_i :\ sim(t, \mathcal{W}_i) > 0\}$.*

The overall spatio-textual score $\tau(p)$ of data object $p$ is defined as in the case of the range score, and the query returns the $k$ objects with the highest score.

The *STDS* algorithm, as defined in Algorithm 1 can be easily adapted for the case of influence score. Only the function *computeScore*$(Q, p)$ must be modified according to the definition of the score variant. Thus, in Algorithm 2 each entry in line 11 will be prioritized according to the influence preference score. In addition, the range restriction is removed in line 5 and line 9. No further modifications are needed, thus in the following we focus on the modifications and optimizations needed for *STPS* algorithm.

**Algorithm 5:** *STPS for influence score*

**Input**: Query $Q$
**Output**: Result set $R$ sorted based on $\tau(p)$

```
1 τ = 0 ;
2 score = −1 ;
3 while (|R| ≤ k) or (best > τ) do
4 |   C = nextCombination(Q) ;
5 |   best = s(C) ;
6 |   R = R∪ getDataObjects(C) ;
7 |   τ = k-th score in R ;
8 return R ;
```

*STPQ* queries based on the influence preference score can be efficiently supported by the *STPS* algorithm with few modifications. Algorithm 5 shows the modified *STPS* for influence preference score. The algorithm continues until at least $k$ data object have been retrieved and until we are sure that none of the remaining data objects can have a better score. We use the score of the $k$-th data object of the current top-$k$ result (line 7) to set a threshold $\tau$. Hence, if the *best* score of any unseen combination is smaller or equal to $\tau$, the algorithm naturally terminates. In more details, $\mathcal{C} = nextCombination(Q)$ is the same with Algorithm 4 and returns the best combination based on score $s(\mathcal{C})$, but without discarding combinations whose distance is greater than $2r$. Thus, in each iteration the combination $\mathcal{C}$ with the highest $\tau(p) = \sum_{i\in[1,c]} \tau_i(p)$ is retrieved. Recall that for the case of the range preference score, all data objects that were located in distance smaller than $r$ from all feature objects of $\mathcal{C}$ had a score equal to $s(\mathcal{C})$. Instead in the case of the influence preference score, $s(\mathcal{C})$ is an upper bound for the score of all data objects based on $\mathcal{C}$. This is because, the computed score is the influence score only for the objects with distance 0, while all other objects have a smaller influence score. Therefore, *getDataObjects*($\mathcal{C}$) must be modified accordingly.

In more details, *getDataObjects*() retrieves the $k$ points that have the highest influence score, by starting a top-$k$ query on the R-Tree (*rtree*) of the data objects. The root is inserted in a heap sorted by the influence score ($\tau(p) = \sum_{i\in[1,c]} \tau_i(p)\dot{2}^{\frac{-dist(p,t_i)}{r}}$). For non-leaf entries $e$ the influence score is computed based on the mindist. Then, the influence score of an entry is an upper bound of any object in the subtree. After retrieving $k$ data objects, we have retrieved the $k$ data objects with the highest influence score for this combination of feature objects. Further improvement can be achieved if *getDataObjects*() stops retrieving data objects based on $\tau$, which reduces the I/Os on *rtree*. If $\tau$ is given to *getDataObjects*() then it will return at most $k$ data objects that have a score smaller than $\tau$. Line 6 merges the results while it removes objects that have been retrieved before. Thus, if an object that is already in the heap is retrieved again the score with the highest value is kept.

After retrieving $k$ data objects with the highest $\tau(p)$ in line 6 (Algorithm 5), the score of the $k$-th data object in $R$ is used as a threshold $\tau$ (line 7). The best score of any unseen combination is $best = s(\mathcal{C})$, which is also an upper bound for the score of any unseen data object, since this is the score for distance 0. Hence, if the *best* score is greater than $\tau$, we have to retrieve additional objects. If the score $s(\mathcal{C})$ of the next combination is smaller than or equal to the threshold we stop retrieving other combinations.

## 7.2 Nearest Neighbor STPQ Queries

In the next score variant, each data object takes as a score the goodness of the feature objects that are its nearest neighbors. In particular, for each feature set the score of the nearest feature object is considered for computing the score of a data object.

DEFINITION 7. *The* ***nearest neighbor preference score*** $\tau_i(p)$ ***of data object*** $p$ *based on the feature set* $F_i$ *is defined as:* $\tau_i(p) = \{s(t) \mid t \in F_i : dist(p,t) \leq dist(p,t') \ \forall t' \in F_i \text{ and } sim(t,\mathcal{W}_i) > 0\}$

The overall spatio-textual score $\tau(p)$ of data object $p$ is defined as in the case of the range score, and the query returns the $k$ objects with the highest score. Again, *STDS* treats nearest neighbor queries similarly as in Algorithm 2 with subtle changes. The range predicate is removed in line 5 and line 9, while the child entries are prioritized in the heap according to their minimum distance from the data object $p$.

Regarding *STPS*, Algorithm 3 is directly applicable for the nearest neighbor score by modifying *nextCombination*($Q$) of Algorithm 4 to return the best combination based on score $s()$, but without discarding combinations that have a $distance > 2r$, as also in the case of the influence score. The remaining challenge is given a combination $\mathcal{C}$ to retrieve the data objects that satisfy the nearest neighbor requirement.

Generally, it is more difficult compared to the other score variants to retrieve the data objects for a given combination $\mathcal{C}$. We need to retrieve all data objects for which the nearest neighbor $t_i$ based on $F_i$ belongs to $\mathcal{C}$. For each feature object $t_i$ of $\mathcal{C}$, there exists a region in which all data points that fall into that region have $t_i$ as their nearest neighbor. This region corresponds to the Voronoi cell [12] and this problem has been studied for finding reverse nearest neighbors [10]. Only the data objects in the intersection of all regions need to be retrieved. In fact, we compute incrementally the Voronoi cell for each feature object $t_i$ of $\mathcal{C}$, which allows us to discard early combinations for which the intersection becomes empty. We omit further implementation details due to space limitations.

# 8. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our algorithms *STDS* and *STPS*, presented previously in Section 5 and Section 6 respectively, for processing spatio-textual preference queries over large disk-resident data. Moreover, we study the gains in performance of our algorithms caused by the SRT index proposed in Section 4 compared to an existing indexing technique ($IR^2$-Tree [8]). In order to ensure a fair comparison, we modify the $IR^2$-Tree to support score values of feature objects. To this end, we add to the leaf nodes of $IR^2$-Tree the scoring values for the feature objects, and maintain in ancestor (internal) nodes the maximum score of all enclosed feature objects. All experiments run on an Intel 2.2GHz processor equipped with 2GB RAM.

## 8.1 Experimental Setup

**Methodology.** In our experimental evaluation, we vary four important parameters of the datasets in order to study

the scalability of the proposed techniques (Section 8.2). These parameters are: (i) the cardinality of the feature sets $|F_i|$, (ii) the cardinality of the set of data objects $|O|$, (iii) the number of feature sets $c$, and (iv) number of distinct keywords indexed. Moreover, we study four different query parameters to study how the characteristics of the query influence the performance of the algorithms (Section 8.3). In more details, we vary (i) the query radius $r$, (ii) the number $k$ of retrieved data objects, (iii) the smoothing parameter $\lambda$ between textual similarity and non-spatial score, and (iv) the number of keywords of the query for each feature set. Finally, we evaluate the performance of *STPS* for the influence score variant (Section 8.4) as well as for the nearest neighbor variant (Section 8.5).

Tested ranges for all parameters are shown in Table 2. The default values are denoted as bold. When we vary one parameter, all others are set to their default values.

| Parameter | Range |
|---|---|
| Cardinality of dataset | $50K$, **100K**, $500K$, $1M$ |
| Cardinality of features sets | $50K$, **100K**, $500K$, $1M$ |
| Number of feature sets $c$ | **2**, 3, 4, 5 |
| Indexed keywords | 64, **128**, 192, 256 |
| Radius $r$ (norm. in $[0, 1]$) | .005, **.01**, .02, .04, .08 |
| $k$ | 5, **10**, 20, 40, 80 |
| Smoothing parameter | .1, .3, **.5**, .7, .9 |
| Queried keywords | 1, **3**, 5, 7, 9 |

**Table 2: Experimental parameters.**

**Datasets.** For evaluating our algorithms, we use both real and synthetic datasets. The real dataset, which was obtained from `factual.com`, describes hotels (≈ 25K objects) and restaurants (≈ 79K objects). In more details we collected restaurant and hotels that are annotated with their location. Moreover, for the collected restaurants we extracted their rating and their textual description of the served food, mentioned as "cuisine". The number of distinct values of keywords for the cuisine is around 130 and each restaurant description may contain one or more keywords. Our datasets contain collected hotels and restaurants for 13 US states that are the states for which `factual.com` lists sufficient data. In addition, we created synthetic clustered datasets of varying size, number of keywords and number of feature sets. Approximately $10,000$ clusters constitute each synthetic dataset. The number of distinct keywords is set to 256 as a default value and each feature object is characterized by one or more keywords that are picked randomly. The spatial constituent of all datasets has been normalized in $[0, 1] \times [0, 1]$. Every reported value is the average of $1,000$ random queries, which are generated in a similar way as the synthetic data and follow the same data distribution.

**Metrics.** The efficiency of all schemes is evaluated according to the average execution time required by a query (time consumed in the CPU and to read disk-pages). In our figures we break down the execution time into the time consumed due to the disk accesses (dark part of the bars) and the time needed for processing the query (CPU time) which is the white part of the bars. The time consumed due to the disk accesses relates to the number of the required I/Os.

## 8.2 Scalability Analysis

In this section, we evaluate the impact of varying different parameters on the efficiency of our algorithms. In order to perform a scalability analysis, we employ the synthetic dataset for this set of experiments. First, we show the scalability limitations of *STDS* for large datasets (Table 3), and then we explore in more detail the significantly superior performance of *STPS*.

| **Feature objects** $\|F_i\|$ | 50000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|
| $IR^2$-tree | 13427.3 | 13854.6 | 25223.1 | 31434.6 |
| SRT | 12301.7 | 13187.9 | 18725.1 | 23046.3 |
| **Data objects** $\|O\|$ | 50000 | 100000 | 500000 | 1000000 |
| $IR^2$-tree | 13073.2 | 13854.6 | 21074.2 | 27846.0 |
| SRT | 11718.1 | 13187.9 | 18267.4 | 23444.9 |
| **Number** $c$ **of** $F_i$ | 2 | 3 | 4 | 5 |
| $IR^2$-tree | 13854.6 | 27842.6 | 33625.0 | 40188.4 |
| SRT | 13187.9 | 14104.9 | 32071.1 | 38340.7 |
| **Indexed keywords** | 1 | 2 | 3 | 4 |
| $IR^2$-tree | 13698.7 | 13854.6 | 15655.6 | 16209.6 |
| SRT | 13121.4 | 13187.9 | 13207.9 | 13887.8 |

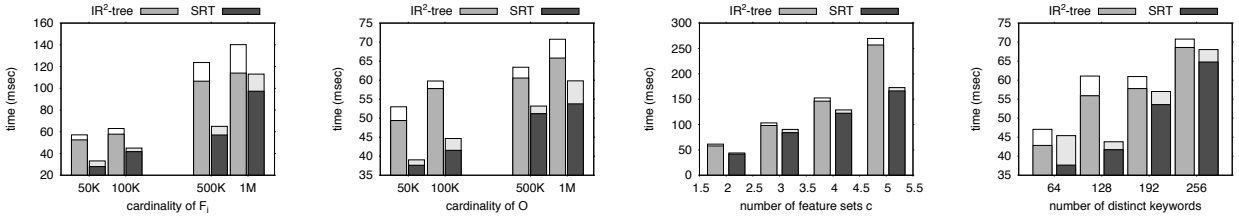
**Table 3: *STDS* execution time (in msec) for synthetic dataset.**

Table 3 shows the results for *STDS* when varying different parameters of the dataset. For the default setting, *STDS* requires over 13 seconds for range queries. Evidently, when a large number of data objects is involved *STDS* does not scale well and the absolute time required is high. The main reason is that *STDS* associates all data objects with $c$ feature objects, which is particularly time-consuming. This experiment demonstrates that a plain algorithm for solving the problem can lead to prohibitive processing cost. Since *STDS* performs badly for all experimental setups, we omit *STDS* for the rest of experimental evaluation, and study the performance of *STPS* coupled with two different indexing techniques.

Figure 7 illustrates the results for the same experiment as above, but for the *STPS* algorithm. We implemented *STPS* over two different indexes: (i) our SRT index (proposed in Section 4), and (ii) the modified $IR^2$-Tree [8] whose nodes are enhanced with the maximum score of enclosed feature objects. In summary, the results clearly demonstrate that *STPS* scales with all parameters and that SRT indexing always outperforms $IR^2$-tree. Moreover, in both cases, the *STPS* algorithm exhibits high performance, as witnessed by the low execution time, which stems from its ability to quickly identify qualified feature combinations. Consequently, the significant gains in processing time (orders of magnitude compared to *STDS*) are mostly due to the effective design of the algorithm. The SRT index additionally offers a speedup of x2 compared to the $IR^2$-Tree, which further improves the overall performance.

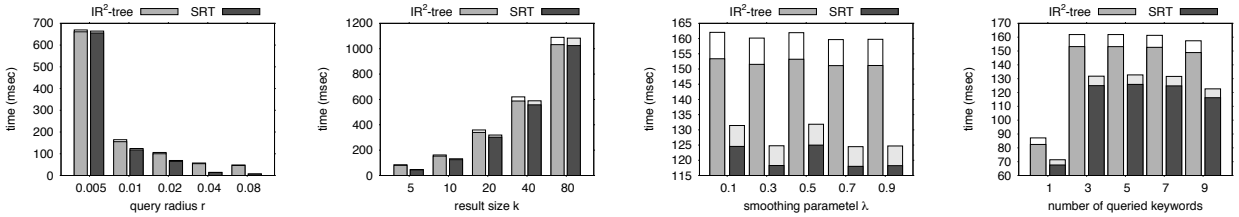
Figure 7(a) shows the execution time when increasing the cardinality of the feature sets. *STPS* scales well since the execution time increases only by a factor of at most x3, when increasing the dataset by one order of magnitude. This increase is due to the increased size of the data structures and the additional processing required to traverse a bigger data structure and find valid combinations of high score. When comparing the index structures, the SRT index is faster, due to the clustering of all score constituents (distance, textual similarity, and non-spatial score) in the 4-dimensional space.

Figure 7(b) shows the obtained results when increasing the number of data objects. Again, *STPS* scales well, and, in fact, even better than in the previous experiment. Obviously, a larger dataset of data objects does not affect the performance so much as larger feature sets. Again, the use




(a) Cardinality of $F_i$ (b) Cardinality of $O$ (c) Varying $c$ (d) Distinct keywords

**Figure 7: Scalability for synthetic dataset.**




(a) Varying r (b) Varying k (c) Varying $\lambda$ (d) Queried keywords

**Figure 8: Range query parameters for real dataset.**

of SRT indexing consistently outperforms the $IR^2$-Tree.

In Figure 7(c), we increase the number of feature categories $c$. As expected, this has a stronger effect on performance, since the cost required to retrieve the highest ranked combinations increases with the number of possible combinations, which, in turn, increases exponentially with $c$. Still, the performance of *STPS* is not severely affected, especially in the case of the SRT index which scales gracefully with $c$.

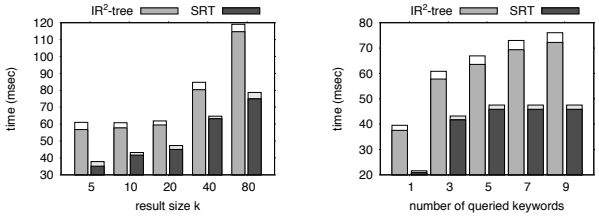
In Figure 7(d), we illustrate how the performance is affected by the number of distinct keywords in the dataset. Apparently, a higher number of keywords causes higher execution times. The reason is twofold. First, as the number of distinct keywords increases, it is less probable to find feature objects that are described by all queried keywords, thus more feature objects need to be retrieved in order to ensure that no other combination has a higher score. Secondly, the node capacity of the index structures drops, thus the height of the index structures may increase, thus causing more IOs. In any case, the increase in the absolute value of execution time is relatively small (20 msec), even when we increase the vocabulary by a factor of 4 (from 64 to 256 keywords).

## 8.3 Varying Query Parameters

In Figure 8, we study the effect of varying query parameters for the real dataset. First, in Figure 8(a), we evaluate the impact of increasing the query radius $r$ on the performance of *STPS*. We notice that for smaller values of $r$ the execution time increases and the gain of SRT indexing compared to $IR^2$-tree drops. For small radius, access to more qualified combinations of feature objects is required, since only few data objects are located in their neighborhood. Therefore, for both indexing approaches the execution time increases mainly due to the increase of the IOs. Since a high percentage of the feature objects need to be retrieved for each feature set, the gain of SRT indexing is small. However, difference in performance becomes obvious for greater values of $r$, and hence, finding relevant feature objects in terms of textual description and good non-spatial score becomes most important for accessing only few feature objects.

Figure 8(b) illustrates the execution time when varying the size of result set $k$. Overall, the execution time increases as $k$ increases. Specifically, with higher values of $k$ more combinations of feature objects are retrieved to compose the result set, which again lead to more IOs to retrieve the qualifying feature objects that constitute valid combinations.

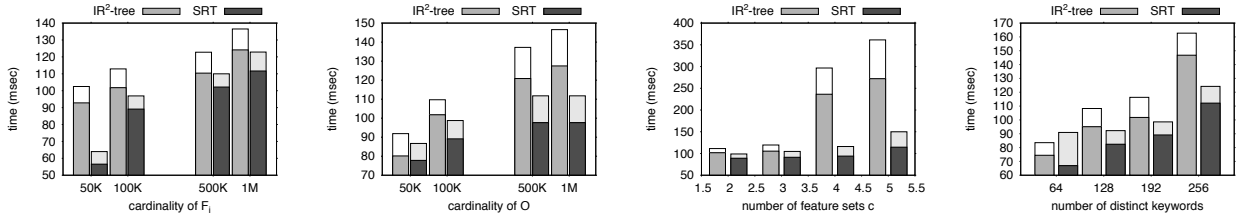
In Figure 8(c), we vary the smoothing parameter $\lambda$. In general, both approaches exhibit relatively stable performance for varying values of $\lambda$. The performance of IR$^2$-tree is not affected by the smoothing parameter, since the feature objects are not grouped into blocks based on the non-spatial score nor based on their textual similarity. We note for the IR$^2$-tree that objects with similar textual descriptions are stored throughout the index, regardless of their non-spatial score; unlike the SRT index where they are clustered together in the same block. As a result, a significant overhead is evident when searching for relevant objects all over the IR$^2$-tree. On the other hand, the SRT index is built by taking into account non-spatial score, the textual information and the spatial location. Thus, *STPS* that uses SRT index is consistently more efficient regardless of the value of the smoothing parameter.




(a) Varying k (b) Queried keywords

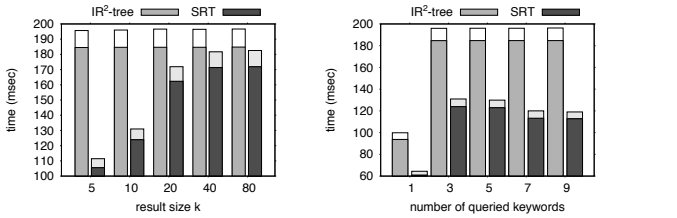
**Figure 9: Query parameters for synthetic dataset.**

In Figure 8(d), we vary the number of queried keywords per feature set from 1 to 9. The number of queried keywords



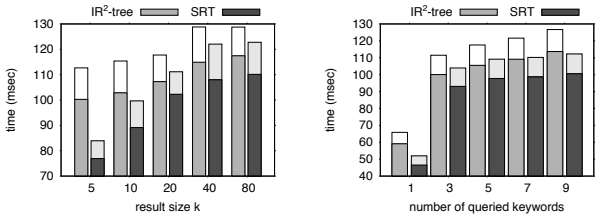

(a) Cardinality of $F_i$ (b) Cardinality of $O$ (c) Varying $c$ (d) Distinct keywords

**Figure 10: Scalability for synthetic dataset and infuence queries.**




(a) Varying k (b) Queried keywords

**Figure 11: Influence query for real dataset.**




(a) Varying k (b) Queried keywords

**Figure 12: Influence query for synthetic dataset.**

has little impact on performance, except for the special case where one keyword is queried for each feature set. This is because both of the indexing techniques aggregate in the non-leaf nodes the textual information of the leaf nodes, which makes it much easier to find objects that contain one keyword, rather than finding objects that are described with more keywords. Nevertheless, the gain in execution time of SRT indexing compared to the $IR^2$-tree is obvious.

Figure 9 depicts results obtained from the synthetic dataset, when varying different query parameters. We notice the same tendency as in the case of the real dataset. In general, we observed that range queries are costlier for the real dataset. This is due to the data distribution: our real dataset, which was extracted from `factual.com`, consists of restaurants and hotels in the US forming just a few clusters. On the other hand, our synthetic dataset is substantially larger and contains a few thousands of clusters. Hence, the data from the latter dataset are more dispersed compared to the former. Last but not least, the SRT indexing consistently outperforms the $IR^2$-tree.

## 8.4 Influence-based Preference Score

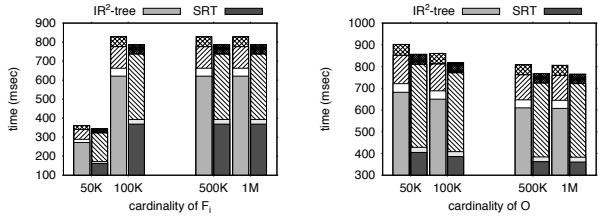
In this section, we study the performance of *STPS* for the influence-based score variant of the spatio-textual preference queries. Figure 10 shows the scalability analysis of *STPS* for this query variant. By comparing the results to Figure 7, which studies the execution time of the range score variant for the same parameters, we conclude that the required execution time is comparable and in some cases slightly increased. This is because more data object for each combination must be retrieved (for the influence-based score variant), since data objects that are further away than $r$ may also have a non-zero score. Nevertheless, the additional cost is not significant in our experiments, and we notice the same tendency in execution time as in the case of range score, thus similar conclusions can be drawn. Moreover, the SRT indexing technique is beneficial in all setups.

Figure 11 shows the execution time of *STPS* for the real dataset when varying query parameters. In Figure 11(a), time decreases for large $k$ values compared to the range score (Figure 8(b)), because combinations with high score are associated with all data objects. Even though the score of the object is reduced based on the distance, still their score is high enough to retrieve fewer combinations. For smaller $k$ values the execution time is not affected significantly. In Figure 11(b), we evaluate the performance of *STPS* when varying the number of queried keywords. We notice that the execution time is similar to Figure 8(d), which depicts the results of the same experiment for range score.

Finally, in Figure 12, we study the performance of *STPS* for the synthetic dataset when varying query parameters. The execution time is similar and slightly higher to the execution time needed for the range score (Figure 9), while the behavior of *STPS* when varying query parameters is the same. Again, the SRT indexing technique improves the performance of *STPS* consistently.

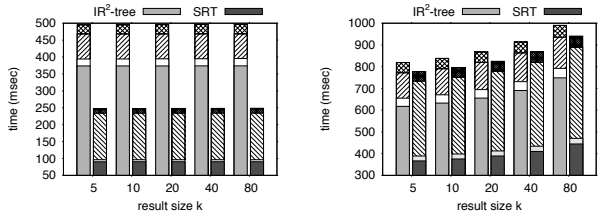
## 8.5 Nearest Neighbor Preference Score

In this section, we evaluate the performance of *STPS* for the nearest neighbor score variant. In general, we noticed that the execution time is higher compared to the other score variants, which is due to the Voronoi cell computations required for retrieving the data objects. In the charts, we illustrate separately with a striped pattern the IO (lower striped part) and the CPU-time (upper striped part) required to compute the respective Voronoi cells. Moreover, it is expected that for a given combination, few data objects satisfy the nearest neighbor constraint, which leads to retrieval of more combinations compared to the other variants. Therefore, we notice in the charts that the execution time is high even if the Voronoi cell computations is not considered (without stripped parts). We note that for static data the Voronoi cells can be pre-computed in a special structure, and therefore significantly reduce the execution time.




(a) Cardinality of $F_i$ (b) Cardinality of $O$

**Figure 13: Scalability of nearest neighbor variant.**




(a) Real dataset (b) Synthetic dataset

**Figure 14: Varying $k$ for nearest neighbor variant.**

Figure 13 depicts the execution time for *STPS* for the synthetic dataset, while varying the size of the feature and object datasets. In Figure 13(a) we notice that for large feature sets the dominant cost is finding the data objects for a given combination (i.e., computing the Voronoi cells), rather than retrieving the combination with the highest score. Computing the Voronoi cells requires retrieval of feature objects from the spatio-textual index of $F_i$ to define the borders of the cell. This cost is higher for the SRT indexing method compared to the $IR^2$-tree, since the $IR^2$-tree is built based on spatial information only and nearby feature objects are stored in the same node. Nevertheless, SRT indexing is still beneficial for *STPS*, but the gain is smaller than for the other variants. Similar conclusions can be drawn when varying the cardinality of the dataset $O$, as depicted in Figure 13(b).

In Figure 14, we vary the parameter $k$ both for real (Figure 14(a)) and synthetic datasets (Figure 14(b)). We notice that the execution time does not increase significantly when increasing $k$ for the real dataset. This is because there exist some combinations for which their feature objects are the nearest neighbor for many data objects. Thus, the same effort is needed for retrieving few or many data objects. This is not the case for the synthetic dataset (Figure 14(b)), where the execution time increases for higher values of $k$.

# 9. CONCLUSIONS

Recently, the database research community has lavished attention on spatio-textual queries that retrieve the objects with the highest spatio-textual similarity to a given query. Differently, in this paper, we address the problem of ranking data objects based on the facilities (feature objects) that are located in their vicinity. A spatio-textual preference score is defined for each feature object that takes into account a non-spatial score and the textual similarity to user-specified keywords, while the score of a data object is defined based on the scores of feature objects located in its neighborhood. Towards this end, we proposed a novel query type called *top-k spatio-textual preference query* and present two query processing algorithms. *Spatio-Textual Data Scan* (*STDS*) first retrieves a data object and then computes its score, whereas *Spatio-Textual Preference Search* (*STPS*) first retrieves highly ranked feature objects and then searches for data objects nearby those feature objects. Moreover, we proposed an indexing technique that improves the performance of our algorithms. Furthermore, we show how our algorithms can support different score variants. Finally, in our experimental evaluation, we put all methods under scrutiny to verify the efficiency and the scalability of our method for processing top-$k$ spatio-textual preference queries.

# Acknowledgments

The work of A. Vlachou was supported by the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

# 10. REFERENCES

[1] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

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

[3] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *Proc. of SIGMOD*, pages 373–384, 2011.

[4] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of interest for user exploration. *PVLDB*, 7(9):733–744, 2014.

[5] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

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

[7] Y. Du, D. Zhang, and T. Xia. The optimal location query. In *Proc. of SSTD*, pages 163–180, 2005.

[8] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *Proc. of ICDE*, pages 656–665, 2008.

[9] I. Kamel and C. Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In *Proc. of VLDB*, pages 500–509, 1994.

[10] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. of SIGMOD*, pages 201–212, 2000.

[11] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *IEEE TKDE*, 23(4):585–599, 2011.

[12] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons Ltd., New York, NY, 2000.

[13] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *Proc. of SSTD*, pages 205–222, 2011.

[14] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2):93–104, 2010.

[15] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proc. of VLDB*, pages 946–957, 2005.

[16] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top-k spatial preference queries. In *Proc. of ICDE*, pages 1076–1085, 2007.

[17] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis. Ranking spatial data by quality preferences. *IEEE TKDE*, 23(3):433–446, 2011.

[18] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *Proc. of ICDE*, pages 688–699, 2009.