RESEARCH-ARTICLE

# A Density-Based Approach to the Retrieval of Top-K Spatial Textual Clusters

**DINGMING WU**, Shenzhen University, Shenzhen, Guangdong, China

**CHRISTIAN S. JENSEN**, Aalborg University, Aalborg, Nordjylland, Denmark



**Published:** 24 October 2016



CIKM'16: ACM Conference on Information and Knowledge Management
*October 24 - 28, 2016*
*Indiana, Indianapolis, USA*

**Conference Sponsors:**
SIGIR
SIGWEB

CIKM '16: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (October 2016)
https://doi.org/10.1145/2983323.2983648
ISBN: 9781450340731

# A Density-Based Approach to the Retrieval of Top-K Spatial Textual Clusters*

Dingming Wu
College of Computer Science & Software Engineering, Shenzhen University
Shenzhen, China
dingming@szu.edu.cn

Christian S. Jensen
Department of Computer Science
Aalborg University
Aalborg, Denmark
csj@cs.aau.dk

## ABSTRACT

Spatial keyword queries retrieve spatial textual objects that are near a query location and are relevant to query keywords. The paper defines the top-$k$ spatial textual clusters ($k$-STC) query that returns the top-$k$ clusters that are located close to a given query location, contain relevant objects with regard to given query keywords, and have an object density that exceeds a given threshold. This query aims to support users who wish to explore nearby regions with many relevant objects. To compute this query, the paper proposes a basic and an advanced algorithm that rely on on-line density-based clustering. An empirical study offers insight into the performance properties of the proposed algorithms.

## 1. INTRODUCTION

Spatial keyword queries [8, 11, 21, 23] allow users to find nearby objects that are relevant to given keywords. For instance, Figure 1(a) illustrates a spatial keyword query $q$ (located at the dot, in London) with keywords 'outdoor seating' that requests the top-5 restaurants (denoted by squares) from TripAdvisor.

(a) Top-5 Objects (b) Top-5 Clusters

**Figure 1: Top-$k$ Objects vs. Top-$k$ Clusters**

While most variants of the spatial keyword query retrieve results with a single-object granularity, we focus on the retrieval of sets of spatially close objects in order to support browsing, or exploratory, user behavior. We view the retrieved sets of objects as spatial textual clusters and propose and study a new type of query, namely the top-$k$ **Spatial Textual Cluster** ($k$-STC) query that returns the top-$k$ clusters such that (i) each cluster contains relevant objects with regard to query keywords, (ii) the density of each cluster satisfies a query constraint, and (iii) the clusters are ranked based on both their spatial distance and text relevance with regards to the query arguments. Figure 1(b) shows an example 5-STC query (black dot) with the same keywords ('outdoor seating') and location as the query in Figure 1(a). The top-5 spatial textual clusters (again restaurants in London) are shown.

We use density-based clustering for finding clusters. The two basic steps to computex the top-$k$ STC query are to obtain the objects that are relevant to the query keywords and to apply clustering to these objects. We consider a cluster scoring function that favors clusters close to the query location and that contain objects with high relevance with regard to the query keywords.

The query keywords are only known when a query arrives, and pre-computing the clusters for all possible sets of query keywords is infeasible. The challenge is then to develop an efficient algorithm that is able to find top-$k$ clusters with a response time that supports interactive search. More specifically, we propose a basic algorithm that combines on-line density-based clustering with an early stop condition. This algorithm applies DBSCAN [9] to objects indexed by an IR-tree [21] to find top-$k$ clusters. In addition, we propose an advanced approach that includes three novel techniques. First, to retrieve clusters, the neighborhood of each object has to be checked in the basic algorithm. We design a skipping rule that reduces the number of objects to be examined. Second, a result cluster consists of dense neighborhoods. However, determining whether a neighborhood is dense in the basic algorithm involves an expansive range query. We design Spatially Gridded Posting Lists (SGPL) to estimate the selectivity of range queries, thus enabling the pruning of sparse neighborhoods. Third, we show how to use the SGPL to gain better range query performance for the range queries that cannot be avoided. An empirical study with real data demonstrates that the paper's proposals offer scalability and are capable of excellent performance.

Section 2 defines the setting. The basic and advanced algorithms are presented in Sections 3 and 4. We report on the empirical performance study in Section 5, and Sections 6 and 7 cover related work and offer conclusions and research directions.

## 2. PROBLEM DEFINITION

We consider a data set $\mathcal{D}$ in which each object $p \in \mathcal{D}$ is a pair $\langle \lambda, \psi \rangle$ of a point location $p.\lambda$ and a text description, or document, $p.\psi$ (e.g., the facilities and menu of a restaurant). Document $p.\psi$ is represented by a vector $(w_1, w_2, \cdots, w_i)$ in which each dimension

*This research was supported in part by project 61502310 from National Natural Science Foundation of China and a grant from the Obel Family Foundation.



*CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA*



DOI: http://dx.doi.org/10.1145/2983323.2983648

corresponds to a distinct term $t_i$ in the document. The weight $w_i$ of a term in the vector can be computed in several different ways, e.g., using tf-idf weighting [16] or language models [15].

We adopt the density-based clustering model [9], and clusters are query-dependent. We revisit key concepts in the context of the top-$k$ spatial textual cluster query.

First, given a set of keywords $\psi$, the **relevant object set** $D_\psi$ satisfies (i) $D_\psi \subseteq \mathcal{D}$ and (ii) $\forall p \in D_\psi (\psi \cap p.\psi \neq \emptyset)$. The **$\epsilon$-neighborhood** of a relevant object $p \in D_\psi$, denoted by $N_\epsilon(p)$, is defined as $N_\epsilon(p) = \{p_i \in D_\psi \mid \|p\ p_i\| \leq \epsilon\}$. An **$\epsilon$-neighborhood** of a relevant object $N_\epsilon(p)$ is **dense** if it contains at least $minpts$ objects, i.e., $|N_\epsilon(p)| \geq minpts$.

Next, several concepts relate to connectedness. A relevant object $p$ is a **core** if its $\epsilon$-neighborhood is dense, and a relevant object $p_i$ is **directly reachable** from a relevant object $p_j$ with regard to $\epsilon$ and $minpts$ if $p_i \in N_\epsilon(p_j)$ and $|N_\epsilon(p_j)| \geq minpts$. A relevant object $p_i$ is **reachable** from a relevant object $p_j$ with regard to $\epsilon$ and $minpts$ if there is a chain of relevant objects $p_1, \cdots, p_n$, where $p_i = p_1$, $p_j = p_n$, such that $p_m$ is directly reachable from $p_{m+1}$ for $1 \leq m < n$. A relevant object $p_i$ is **connected** to a relevant object $p_j$ with regard to $\epsilon$ and $minpts$ if there is a relevant object $p_m$ such that both $p_i$ and $p_j$ are reachable from $p_m$ with regard to $\epsilon$ and $minpts$.

A **spatial textual cluster** $R$ with regard to $\psi, \epsilon$, and $minpts$ is a subset of $D_\psi$ and is a maximal set such that $\forall p_i, p_j \in R$, $p_i$ and $p_j$ are connected through dense $\epsilon$-neighborhoods when considering only objects in $D_\psi$.

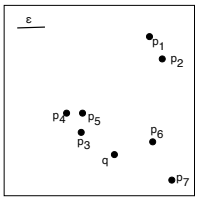
A **top-$k$ Spatial Textual Cluster** ($k$-STC) query $q = \langle \lambda, \psi, k, \epsilon, minpts \rangle$ takes five arguments: a point location $\lambda$, a set of keywords $\psi$, a number of requested object sets $k$, a distance constraint $\epsilon$ on neighborhoods, and the minimum number of objects $minpts$ in a dense $\epsilon$-neighborhood. It returns a list of $k$ spatial textual clusters that minimize a scoring function and that are in ascending order of their scores. The maximality of each cluster implies that the top-$k$ clusters do not overlap. The density requirement parameters $\epsilon$ and $minpts$ are able to capture how far the user is willing to move before reaching another object.

Intuitively, a cluster with high text relevance and that is located close to the query location should be given a high ranking in the result. We thus use the following scoring function.

$$score_q(R) = \alpha \cdot d_{q.\lambda}(R) + (1 - \alpha) \cdot (1 - tr_{q.\psi}(R)), \quad (1)$$

where $d_{q.\lambda}(R)$ is the minimum spatial distance between the query location and the objects in $R$ and $tr_{q.\psi}(R)$ is the maximum text relevance in $R$. The approaches we present are applicable to scoring functions that are monotone with respect to both spatial distance and text relevance. Parameter $\alpha$ is used to balance the spatial proximity and the text relevance of the retrieved clusters. All spatial distances and text relevances are normalized to $[0, 1]$.

**Example 2.1:** Consider the example $k$-STC query $q$ with location $q.\lambda$ and $q.\epsilon$ as shown in Figure 2) and with $q.\psi = \{coffee, tea\}$, $q.k = 1$, and $q.minpts = 2$. The data set contains the 7 objects $p_1, p_2, \cdots, p_7$. The figure also shows the document vector and the Euclidean distance to the query location for each object. Let $\alpha = 0.5$ and $tr_{q.\psi}(p.\psi) = \sum_{t \in q.\psi \cap p.\psi} w_t$. The top-1 cluster is $R = \{p_3, p_5\}$ that has score 0.315 ($= 0.5 \times (0.11 + 0.15)/2 + (1 - 0.5) \times (1 - (0.5 + 0.5)/2)$). □




| | coffee | tea | pizza | $d_{q.\lambda}(p_i.\lambda)$ |
|---|---|---|---|---|
| $p_1$ | (0.2, | 0.2, | 0) | 0.25 |
| $p_2$ | (0.2, | 0.2, | 0) | 0.2 |
| $p_3$ | (0.5, | 0, | 0) | 0.11 |
| $p_4$ | (0, | 0, | 0.5) | 0.18 |
| $p_5$ | (0, | 0.5, | 0) | 0.15 |
| $p_6$ | (0.5, | 0, | 0) | 0.1 |
| $p_7$ | (0.5, | 0.5, | 0) | 0.19 |

**Figure 2: Example $k$-STC Query**

## 3. BASIC APPROACH

We adopt the IR-tree [21] and inverted file [24] index structures to organize objects. An inverted file index has two main components: (i) A vocabulary of all distinct words appearing in the text descriptions of the objects in the data set and (ii) a posting list for each word $t$, i.e., a sequence of pairs $(id, w)$, where $id$ is the identifier of an object whose text description contains $t$ and $w$ is the word's weight in the object.

The IR-tree extends the R-tree with inverted files. Each leaf node contains entries of the form $e_0 = (id, \Lambda)$, where $e_0.id$ refers to an object identifier and $e_0.\Lambda$ is a minimum bounding rectangle (MBR) of the location of the object. Each leaf node also contains a pointer to an inverted file indexing the text descriptions of all objects stored in the node. Each non-leaf node $N$ contains entries of the form $e = (id, \Lambda)$, where $e.id$ points to a child node of $N$ and $e.\Lambda$ is the MBR of all rectangles in entries of the child node. Each non-leaf node also contains a pointer to an inverted file indexing the pseudo text descriptions of the entries stored in the node. A pseudo text description of an entry $e$ is a summary of all (pseudo) text descriptions in the entries of the child node pointed to by $e$. This enables derivation of an upper bound on the text relevance to a query of any object contained in the subtree rooted at $e$.

Given a query, the top-$k$ spatial textual clusters are the top-$k$ density-based clusters found from the relevant object set $D_\psi$ with regard to the query keywords. A straightforward solution is (i) to obtain the object set $D_\psi$, (ii) to find all density-based clusters in $D_\psi$, (iii) to sort these according to the scoring function, and (iv) to return the top-$k$ clusters. This solution is inefficient, since finding all clusters is expensive. The basic algorithm avoids finding all clusters. Specifically, some candidate clusters are first obtained, and a threshold is set according to the score of the $k$-th candidate cluster. The basic algorithm estimates a bound on the scores of all unfound clusters. If the bound is worse than the threshold, the currently found top-$k$ clusters are the result.

As shown in Algorithm 1, the basic algorithm first obtains the relevant object set $D_\psi$ with regard to the query keywords by taking the union of the posting lists of the query terms in the inverted index (line 1). Next, it sorts the objects in $D_\psi$ in ascending order of their Euclidean distances to the query location, i.e., $d_{q.\lambda}(p.\lambda)$, and keeps the sorted copy i *slist* (line 2). In addition, the objects in $D_\psi$ are sorted in ascending order of their converted text relevance with regard to the query keywords, i.e., $1 - tr_{q.\psi}(p.\psi)$, and the sorted copy is kept in *tlist* (line 3).

The candidate list *rlist* is initialized as empty (line 4). The threshold is set to infinity (line 5). The algorithm does sorted access in parallel on the sorted lists *slist* and *tlist* (line 7). As an object $p$ is obtained from one of the lists, function **GetCluster** (Algorithm 2) tries to retrieve the cluster containing $p$ as a core object (line 8). Meanwhile, the objects contained in the cluster are removed from both *slist* and *tlist*. If the retrieved cluster is not empty, it is added to the candidate list *rlist*, and the threshold $\tau$ is updated to the score of the $k$-th candidate in *rlist* (lines 9–11). The algorithm estimates a lower bound on the scores of all unfound clusters by using the minimal Euclidean distance $sb$ and minimal converted text relevance $tb$ of all objects left in *slist* and *tlist*, i.e., $bound = \alpha \cdot sb + (1 - \alpha) \cdot tb$ (lines 12–14). The result is guaranteed to be found when $bound \geq \tau$ or *slist* is exhausted (indicating that *tlist* is also exhausted) (line 15). The top-$k$ candidate clusters in *rlist* are the result.

LEMMA 1. *The stop condition (line 15) in Algorithm 1 guarantees the correct result.*

PROOF. Omitted. □

**Algorithm 1** **Basic**(Query $q$, IR-tree $irtree$, InvertedIndex $iindex$, Integer $k$)

```
1: D_ψ ← LoadRelevantObjects(q.ψ, iindex);
2: slist ← sort objects in D_ψ in ascending order of d_{q.λ}(p.λ);
3: tlist ← sort objects in D_ψ in ascending order of 1 − tr_{q.ψ}(p.ψ);
4: rlist ← ∅;
5: τ ← ∞;
6: repeat
7:     Object p ← sorted access in parallel to slist and tlist;
8:     c ← GetCluster(p, q, irtree, tlist, slist);
9:     if c ≠ ∅ then
10:        Add c to rlist;
11:        τ ← score of the k-th cluster in rlist;
12:    sb ← First(slist);
13:    tb ← First(tlist);
14:    bound ← α · sb + (1 − α) · tb;
15: until bound ≥ τ ∨ slist = ∅
16: Return rlist;
```

To retrieve a cluster $R$ containing $p$ as a core object, function **GetCluster** (Algorithm 2) issues a range query centered at $p$ with radius $q.\epsilon$ on the IR-tree (line 2). The goal is to check whether the $\epsilon$-neighborhood of $p$ is dense. If $neighbors$ contains fewer than $q.minpts$ objects, object $p$ is marked as noise, and an empty set is returned (lines 3–6). Otherwise, $neighbors$ is considered as a temporary cluster (line 8) that is expanded by checking the $\epsilon$-neighborhood of each object $p_i$ in $neighbors$ except $p$ (lines 12 and 13). If the $\epsilon$-neighborhood of $p_i$ is dense (line 14), the objects inside and previously labeled as noise are added to the temporary cluster (lines 16 and 17). Next, the objects inside and not belonging to the temporary cluster are added to both the temporary cluster and to $neighbors$ in preparation for further expansion in subsequent iterations (lines 18–21). To avoid duplicate operations, the objects that are marked as noise or are added to the temporary cluster are removed from $tlist$ and $slist$ (lines 4, 9, 20). The temporary cluster $R$ is finalized and returned if no more object can be added.

Function **RangeQuery** (Algorithm 3) is used to find the objects in the $\epsilon$-neighborhood of an object $p$ using an IR-tree on the objects. A priority queue organizes the IR-tree nodes to be visited using the minimum Euclidean distance between nodes and the query as the key. First, the root node of the IR-tree is added to the queue (line 3). The algorithm iteratively visits and removes the first node in the queue (lines 5 and 6). If the node is a leaf node, the objects in it that are relevant to the query keywords and that are located in range $q.\epsilon$ are added to $neighbors$ (lines 7–10). Otherwise, the node is a non-leaf node. Its child nodes that are relevant to the query keywords and that are located in range $q.\epsilon$ are inserted into the priority queue (lines 12–14). The process terminates when the queue is exhausted.

# 4. ADVANCED APPROACH

The basic approach is inefficient because it checks the neighborhoods of all relevant objects with regard to the query keywords and because checking a neighborhood involves a time-consuming range query. The advanced approach includes three techniques that address these inefficiencies.

## 4.1 Object Skipping

Function **GetCluster** tries to find a cluster $R$ containing a relevant object $p$ as a core object. It first determines whether the $\epsilon$-neighborhood of $p$ is dense. If so, cluster $R$ is initialized as the relevant objects inside the$p$'s $\epsilon$-neighborhood. An object is examined if its neighborhood has been checked. Next, the relevant objects other than $p$ in the $\epsilon$-neighborhood of $p$ are examined in turn. If a neighborhood under consideration is dense, the newly found relevant objects in it are added to $R$. This way, $R$ is finalized when each relevant object has been examined. However, it is possible to get a cluster $R$ by examining only a portion of the relevant objects. Consider Figure 3, where the neighborhoods of objects $p_1$, $p_2$, $p_3$, and $p_4$ are given as dashed and solid circles. The neighborhood of $p_4$ (solid circle) is covered by the union of the neighborhoods of $p_1$, $p_2$, and $p_3$ (dashed circles), making it unnecessary to check the neighborhood of $p_4$ after having examined $p_1$, $p_2$, and $p_3$. Based on this observation, we define a skipping rule that reduces the number of relevant objects to be examined, and we design an algorithm **OS** that implements the rule.

**Algorithm 2** **GetCluster**(Object $p$, Query $q$, IR-tree $irtree$, List $tlist$, List $slist$)

```
1: R ← ∅;
2: neighbors ← irtree.RangeQuery(q, p);
3: if neighbors.size < q.minpts then
4:     Remove p from tlist and slist;
5:     Mark p as a noise;
6:     Return R;
7: else                                          ▷ p is a core;
8:     Add neighbors to R;
9:     Remove neighbors from tlist and slist;
10:    Remove p from neighbors;
11:    while neighbors is not empty do
12:        Object p_i ← remove an object from neighbors;
13:        neighbors_i ← irtree.RangeQuery(q, p_i);
14:        if neighbors_i.size ≥ q.minpts then
15:            for each object p_j in neighbors_i do
16:                if p_j is a noise then
17:                    Add p_j to R;
18:                else if p_j ∉ R then
19:                    Add p_j to R;
20:                    Remove p_j from tlist and slist;
21:                    Add p_j to neighbors;
22:    Return R;
```

**Algorithm 3** **RangeQuery**(Query $q$, Object $p$)

```
1: neighbors ← ∅;
2: PriorityQueue queue ← ∅;
3: queue.Enqueue(root);
4: while queue is not empty do
5:     e ← queue.Dequeue();
6:     N ← ReadNode(e);
7:     if N is a leaf node then
8:         for each object o in N do
9:             if o is relevant to q.ψ and ||p o||_min ≤ q.ϵ then
10:                neighbors.Add(o);
11:    else
12:        for each entry e′ in N do
13:            if e′ is relevant to q.ψ and ||p e′||_min ≤ q.ϵ then
14:                queue.Enqueue(e′);
15: Return neighbors;
```

SKIPPING RULE 1. *Let $S = (p_1, p_2, \cdots, p_n)$ be the order in which a set of objects is examined. Object $p_i$ $(i > 1)$ can be skipped if the neighborhood of $p_i$ is fully covered by the union of the neighborhoods of the objects examined before $p_i$, i.e., if $N_\epsilon(p_i) \subset \cup_{1 \le j < i} N_\epsilon(p_j)$, where $N_\epsilon(p_i)$ is represented as a circular region centered at $p_i$ and with radius $\epsilon$.*

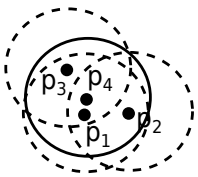



**Figure 3: Neighborhoods**

The skipping rule is effective when given a good ordering $S$. In Figure 3, if $S = (p_1, p_2, p_4, p_3)$, no object can be skipped. However, if $S = (p_1, p_2, p_3, p_4)$, object $p_4$ can be skipped. Intuitively, if the union of the neighborhoods of the objects that have been examined covers a large area, the probability of skipping the next object is high.

To achieve an algorithm **OS** that implements the skipping rule, Algorithm 2 is modified in two ways. First, given an object $p$, the objects in $p$'s neighborhood are sorted in descending order of their distance to $p$. The idea is that the farther the objects are from $p$, the larger the area covered by their neighborhoods is. Referring to lines 2 and 13 in Algorithm 2, the objects returned from **RangeQuery** are sorted in descending order of their distances. Let $S(p)$ be the sorted list of the objects in the neighborhood of $p$. List *neighbors* (line 2 in Algorithm 2) maintains the objects to be examined. It is initialized as the sorted list $S(p)$. For each object $p_i$ in $S(p)$, the sorted list $S(p_i)$ of $p_i$ is appended to $S(p)$. The algorithm terminates when $S(p)$ is exhausted. Second, each time, when about to check the neighborhood of an object (line 13 in Algorithm 2), the skipping rule is considered. If the rule applies, the algorithm continues to process the next object. The skipping rule involves the testing of whether a circle is covered by the union of several circles. This can be accomplished using a recursive subdivision of the circle by non-overlapping squares.

## 4.2 Spatially Gridded Posting Lists

We proceed to design **spatially gridded posting lists** (SGPL) to estimate the selectivity of a range query on the IR-tree, such that sparse neighborhoods can be pruned without issuing expensive range queries.

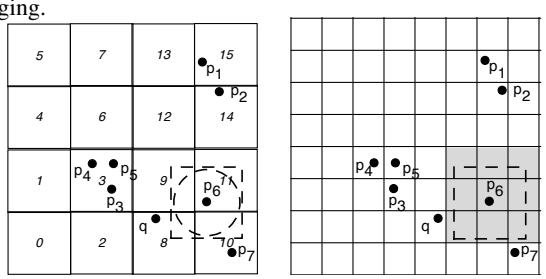
An $n \times n$ grid is created on the data set. For each word $w$, a spatially gridded posting list is constructed covering all the objects that contain $w$. Let $D_{w_i}$ be the set of objects containing word $w_i$. The SGPL of $w_i$ is a sorted list of entries, where each entry takes the form $(c_j, S_{w_i,c_j})$, where $c_j$ (sorting key) is the index value of a grid cell $C_{c_j}$ and $S_{w_i,c_j}$ is a set of objects that belong to $D_{w_i}$ and that are located in grid cell $C_{c_j}$, i.e., $\forall p \in S_{w_i,c_j}(p \in D_{w_i} \wedge p.\lambda \in C_{c_j})$. Grid cells are indexed using a space filling curve, e.g., a Hilbert curve or a Z-order curve. The SGPLs of all distinct words in the data set are organized similarly to the inverted file. Empty cells are not stored. Given a word, its SGPL can be retrieved straightforwardly.

**Example 4.1:** Figure 4(a) illustrates a $4 \times 4$ grid on the 7 objects in Example 2.1. Grid cells are indexed using a 2-order Z-curve. Numbers in italics are the Z-order derived keys for the cells. Figure 4(c) shows the SGPLs for words 'coffee' and 'tea'. For example, the entry for 'coffee' tells that the document of $p_3$ contains 'coffee' and that $p_3$ is located in the cell with index value 3. □

Given a set $q.\psi$ containing $m$ query keywords, the corresponding $m$ SGPLs are merged to estimate the selectivity of the query. We define a merging operator $\bigoplus$ on SGPLs that produces a count for each non-empty cell. The count for cell $C$ is the cardinality of the union of the sets of objects located in $C$ from different SGPLs, i.e.,

$$\bigoplus_{w_i \in q.\psi} SGPL_{w_i} = \{(c_j, |\bigcup_{w_i \in q.\psi} S_{w_i,c_j}|\}. \quad (2)$$

**Example 4.2:** The third row in Figure 4(c) is the result of merging the SGPLs of words 'coffee' and 'tea'. For example, the entry $(3, 2)$ tells that the cell with index value 3 contains 2 objects after merging. □




(a) $4 \times 4$ Grid (b) $8 \times 8$ Grid

| | |
|---|---|
| coffee | $(3, \{p_3\}), (10, \{p_7\}), (11, \{p_6\}), (14, \{p_2\}), (15, \{p_1\})$ |
| tea | $(3, \{p_5\}), (10, \{p_7\}), (14, \{p_2\}), (15, \{p_1\})$ |
| coffee $\bigoplus$ tea | $(3, 2), (10, 1), (11, 1), (14, 1), (15, 1)$ |

(c) SGPL

**Figure 4: Example Spatially Gridded Posting Lists**

The merged result of the SGPLs of the query keywords is used to estimate the selectivity of the circular range query $q_c$ centered at an object $p$ with radius $\epsilon$ (e.g., the dashed circle in Figure 4(a)). We approximate the circle $q_c$ as its circumscribed square $q_s$ (e.g., the dashed square in Figure 4(a)). The sum of the counts of the grid cells that intersect square $q_s$ in the merged SGPLs of the query keywords is returned as the selectivity. It is not necessary to merge the whole SGPL of each query keyword—only the cells that intersect $q_s$ need to be considered. We thus define a parameterized merging operator $\bigoplus(q_s)$ as follows.

$$\bigoplus_{w_i \in q.\psi} (q_s) SGPL_{w_i} = \{(c_j, |\bigcup_{w_i \in q.\psi} S_{w_i,c_j}| \mid C_{c_j} \cap q_s \neq \emptyset\} \quad (3)$$

In Figure 4, where $q_s$ is the dashed square, $coffee \bigoplus(q_s) tea = \{(10, 1), (11, 1)\}$. Based on the coding scheme of the space-filling curve, we adopt an efficient existing algorithm [10] to retrieve the cells that intersect the query range $q_s$.

The derived selectivity serves as an upper bound on the number of objects falling inside circle $q_c$. If the selectivity (upper bound) is less than $q.minpts$, the $\epsilon$-neighborhood of object $p$ is guaranteed to be sparse. Otherwise, function **RangeQuery** is called to compute the exact number of objects in the $\epsilon$-neighborhood of object $p$. Hence, some sparse $\epsilon$-neighborhoods can be found efficiently. However, it is not guaranteed that all sparse relevant $\epsilon$-neighborhoods can be found using the SGPLs of query keywords, since the selectivity is not always a tight upper bound due to the granularity of the grid and the use of the circumscribed square of the circular range.

**Example 4.3:** Let $q.minpts = 2$objects. In Figure 4(a), the dashed square $q_s$ approximates the $\epsilon$-neighborhood of $p_6$. Since $q_s$ intersects grid cells 8, 9, 10, and 11 in the merged result of the SGPLs of words 'coffee' and 'tea', the selectivity is 2, i.e., the number of objects covered by the four grid cells. Hence, the $\epsilon$-neighborhoods of $p_6$ is conservatively assessed as not sparse, and **RangeQuery** is called to compute the exact number of objects in the $\epsilon$-neighborhood. However, if using the finer grid in Figure 4(b), the selectivity is 1 ($< q.minpts$). Only the gray cells intersect the

query range. Thus, the $\epsilon$-neighborhood of $p_6$ is guaranteed to be sparse, with no need to call function $\mathbf{RangeQuery}$. □

We observe that using a finer grid may help avoid expensive $\mathbf{RangeQuery}$ operations. However, the cost of the selectivity estimation increases when using a finer grid. The empirical study covers the effects of the grid granularity.

## 4.3 FastRange

SGPLs can also be used to support the processing of range queries on the IR-tree. We propose an algorithm $\mathbf{FastRange}$ that handles range queries on SGPLs. We first override the merging operator $\bigoplus(q_s)$ as $\overline{\bigoplus}(q_s)$.

$$\overline{\bigoplus_{w_i \in q.\psi}}(q_s) SGPL_{w_i} = \{(c_j, \bigcup_{w_i \in q.\psi} S_{w_i,c_j} \mid C_{c_j} \cap q_s \neq \emptyset\} \quad (4)$$

Operator $\bigoplus(q_s)$ computes the *number* of objects inside each cell intersecting query range $q_s$, while operator $\overline{\bigoplus}(q_s)$ computes the *set of the identifiers* of the objects inside each such cell. As shown in Algorithm 4, $\mathbf{FastRange}$ takes two arguments: $list$, the result of operator $\overline{\bigoplus}(q_s)$, and $q_c$, the circular region centered at an object $p$ and with radius $\epsilon$. If a cell $c$ from $list$ is contained in the query range $q_c$, all the objects in $c$ are added to the result (lines 3 and 4). If a cell $c$ intersects $q_c$, only objects in $c$ that have distance to $p$ no greater than $\epsilon$ are added to the result (lines 6–8).

**Algorithm 4** $\mathbf{FastRange}$(SGPL $list$, Range $q_c$)

```
1: result ← ∅;
2: for each cell c ∈ list do
3:     if c is completely inside the query range q_c then
4:         All the objects inside c are added to result
5:     else                                  ▷ c intersects the query range
6:         for each object o inside c do
7:             if ||o p|| ≤ ε then
8:                 Add o to result;
```

# 5. EMPIRICAL STUDIES

**Experimental Setup.** We use a real data set from TripAdvisor that contains 100,789 restaurants. Each restaurant has a text description of length 3 words on average. The total number of distinct words is 202. The data set is small. But as we are not aware of other public real data sets that match our problem motivation, we generate larger data sets for scalability evaluation. Specifically, we generate data sets of size 200K, 400K, 600K, 800K, and 1M. For a randomly picked object $p$ in TripAdvisor, we generate a new object whose location is obtained by slightly shifting the location of $p$ and whose text description is the same as that of $p$.

We generate 4 query sets with 1, 2, 3, and 4 keywords, respectively. Each set comprises 100 queries. No query has an empty result. To generate a query, we randomly pick an object in the dataset and use its location as the query location, and we randomly choose words from the object as the query keywords.

We evaluate the performance of the basic approach (Basic), the advanced approach with object skipping (Adv1), the advanced approach with object skipping and selectivity estimation (Adv2), and the advanced approach with object skipping, selectivity estimation, and FastRange (Adv3), under different parameter settings. Table 1 shows the parameter values used in the experiments, where the bold values are default values. All algorithms were implemented in Java, and an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 16GB main memory was used for the experiments. All the data structures are memory resident.

**Table 1: Parameter Values**

| Interpretation | Parameter | Values |
|---|---|---|
| # of clusters | $k$ | 5, **10**, 15, 20 |
| # of query keywords | $\|q.\psi\|$ | 1, **2**, 3, 4 |
| Density requirements | $\epsilon$ | 0.0001, 0.0005, **0.001**, 0.005, 0.01 |
| | $minpts$ | 10, 20, **50**, 100, 200 |
| Z-curve order, SGPL | $h$ | 3, 4, 5, **6**, 7, 8, 9, 10 |
| Spatial/textual weight | $\alpha$ | 0.1, 0.3, **0.5**, 0.7, 0.9 |
| Data set size | $\|\mathcal{D}\|$ | **100K**, 200K, 400K, 600K, 800K, 1M |

**Varying the Number of Keywords $|q.\psi|$.** Figure 5 shows how the four approaches are affected by the number of query keywords. The advanced algorithms exhibits the best performance. The number of range queries issued and the elapsed time are aligned. In subsequent experiments, we do not show the number of range queries.

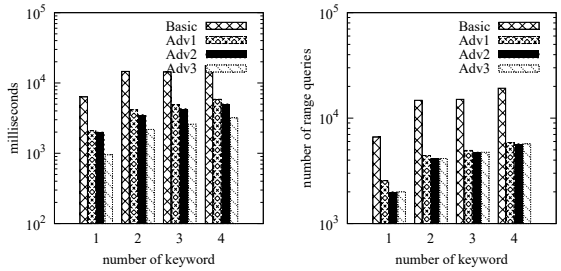



**Figure 5: Varying the Number of Keywords**

**Varying Density Requirement.** Figure 6 shows the effects of varying $\epsilon$ and $minpts$. As $\epsilon$ increases, more object neighborhoods become dense, and if the $\epsilon$-neighborhood of a relevant object satisfies the density requirement, we need to examine the relevant objects inside the $\epsilon$-neighborhood to expand the cluster. Hence, the cost increases. However, the performance is not sensitive to $minpts$ in range $[10, 200]$ given that $\epsilon = 0.001$.

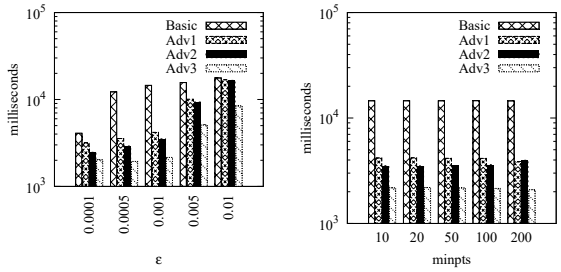



**Figure 6: Varying $\epsilon$ and $minpts$**

**Varying $\alpha$ and the Number $k$ of Requested Clusters.** The four approaches are insensitive to varying $\alpha$ in the range 0.1 to 0.9 and to varying the number $k$ of requested clusters in the range 5 to 20.

**Scalability.** Figure 7 shows the effect of varying the data set size. We observe that the algorithms scale linearly and that the advanced algorithm is affected the least by an increasing data set size.

**Varying the Order of Z-Curve $h$ in SGPL.** SGPL uses a grid indexed by a Z-curve. The order $h$ of the Z-curve defines the granularity of the grid: a larger $h$ yields a finer grid. A finer grid improves the ability to estimate the selectivity of range queries and the ability to detect sparse $\epsilon$-neighborhoods. Figure 7 shows that as $h$ increases, the performance improves. When $h$ exceeds 6, the performance of SGPL becomes stable, suggesting that the selectivity estimation ability is close to optimal at this point.

**Summary.** Overall, the best advanced approach outperforms the basic approach by an order of magnitude. The elapsed time and the number of range queries are proportional to each other, which indicates that the time consuming part of the $k$-STC queries is the range

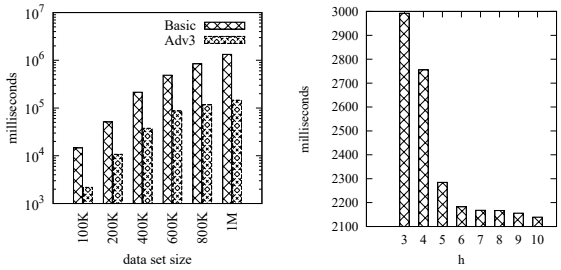



**Figure 7: Varying Data Size and the Order $h$ in SGPL**

queries. The proposed advanced approach significantly reduce the cost incurred by the range queries.

# 6. RELATED WORK

Several studies consider queries that are similar to that of this study. One study [2, 18] targets the case where the user wishes to find nearby groups of relevant objects. Considering the co-location relationship between objects, other studies [7, 12, 19] consider the retrieval of regions so that the total weights of objects inside the regions are maximized. In these studies, result regions are either fixed-sized rectangles or circles. A recent study [6] computes a maximum-sum region where the road network distance between objects is less than a query constraint and the sum of the scores of the objects inside the region is maximized. This kind of query may retrieve a region containing many objects with low scores and ignore a promising region with few objects with high scores. The collective spatial keyword query [3, 5, 14] finds a group of objects that are close to a query point and that collectively cover a set of query keywords. Another query [4] uses prestige-based relevance for ranking. None of the above studies use clustering techniques, and we believe that the paper's study is the first to attempt to use clustering for spatial keyword querying.

Next, the most popular density based clustering method is DBSCAN [9]. Its "density-reachability" connects objects within a certain distance threshold of each other, and a cluster consists of all density-connected objects together with all objects that are within the threshold distance of these objects. Such clusters can have arbitrary shapes. OPTICS [1] generalizes DBSCAN, removing the need to choose a value for the distance threshold. DBRS [20] improves DBSCAN by repeatedly picking an unclassified point at random and examining its neighborhood. VDBSCAN [13] aims for varied-density dataset analysis. GDBSCAN [17] clusters objects according to their spatial and their non-spatial attributes. While the paper's study uses DBSCAN, it is possible to also use other density-based clustering models that are based on DBSCAN.

An extended version of this paper is available [22].

# 7. CONCLUSIONS AND FUTURE WORK

This paper proposes a new type of query, namely the top-$k$ spatial textual clusters ($k$-STC) query that returns the top-$k$ clusters that (i) are located the closest to the query location, that (ii) contain the objects whose text descriptions are the most relevant to the query keywords, and that (iii) have densities that exceed a threshold. Qualifying clusters are ranked according to a scoring function that takes into account the distance to the query location and the relevance to the query keywords. We propose a basic algorithm that is an on-line clustering method. To improve performance, we propose an advanced approach that includes three techniques: (i) a skipping rule that is used to reduce the number of objects to be examined, (ii) spatially gridded posting lists (SGPL) that are used to estimate the selectivity of the range queries so that sparse neighborhoods can be pruned at low cost, and (iii) a fast range query algorithm that leverages the SGPL. Empirical studies on a real data set indicate that the paper's proposals are capable of excellent performance.

This work opens to a number of interesting research directions. For example, the density requirements $\epsilon$ and $minpts$ in the query define the form of the clusters to be retrieved. However, a density requirement good for a city center may not work for the countryside. Thus, it is of interest to enable parameters to vary according to the local data density.

# 8. REFERENCES


[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. In *SIGMOD*, pages 49–60, 1999.

[2] K. Bøgh, A. Skovsgaard, and C. S. Jensen. Groupfinder: A new approach to top-k point-of-interest group retrieval. *PVLDB*, 6(12):1226–1229, 2013.

[3] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *ACM TODS*, 40(2):13, 2015.

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

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

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

[7] D.-W. Choi, C.-W. Chung, and Y. Tao. A scalable algorithm for maximizing range sum in spatial databases. *PVLDB*, 5(11):1088–1099, 2012.

[8] 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.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[10] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. In *BNCOD*, pages 20–35, 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] J. Liu, G. Yu, and H. Sun. Subject-oriented top-k hot region queries in spatial dataset. In *CIKM*, pages 2409–2412, 2011.

[13] P. Liu, D. Zhou, and N. Wu. VDBSCAN: Varied density based spatial clustering of applications with noise. In *Service Systems and Service Management*, pages 1–4, 2007.

[14] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, pages 689–700, 2013.

[15] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, 1998.

[16] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, 1975.

[17] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Min. Knowl. Discov.*, 2(2):169–194, 1998.

[18] A. Skovsgaard and C. S. Jensen. Finding top-$k$ relevant groups of spatial web objects. *VLDB J.*, 24(4):537–555, 2015.

[19] Y. Tao, X. Hu, D.-W. Choi, and C.-W. Chung. Approximate maxrs in spatial databases. *PVLDB*, 6(13):1546–1557, 2013.

[20] X. Wang and H. J. Hamilton. DBRS: A density-based spatial clustering method with random sampling. In *PAKDD*, pages 563–575, 2003.

[21] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *VLDB J.*, 21(6):797–822, 2012.

[22] D. Wu and C. S. Jensen. A density-based approach to the retrieval of top-k spatial textual clusters. *CoRR*, abs/1607.08681, 2016.

[23] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *IEEE TKDE*, 24(10):1889–1903, 2012.

[24] J. Zobel and A. Moffat. Inverted files for text search engines. In *ACM Comput. Surv.*, volume 38, article 6, 2006.