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

Keyword-based web queries with local intent retrieve web content that is relevant to supplied keywords and that represent points of interest that are near the query location. Two broad categories of such queries exist. The first encompasses queries that retrieve single spatial web objects that each satisfy the query arguments. Most proposals belong to this category. The second category, to which this paper's proposal belongs, encompasses queries that support exploratory user behavior and retrieve sets of objects that represent regions of space that may be of interest to the user. Specifically, the 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 a given query location, (ii) contain the most relevant objects with regard to given query keywords, and (iii) have an object density that exceeds a given threshold. To compute this query, we propose a basic algorithm that relies on on-line density-based clustering and exploits an early stop condition. To improve the response time, we design an advanced approach that includes three techniques: (i) an object skipping rule, (ii) spatially gridded posting lists, and (iii) a fast range query algorithm. An empirical study on real data demonstrates that the paper's proposals offer scalability and are capable of excellent performance.

1. INTRODUCTION

Spatial keyword query processing [10,12,15,17,18,21,25,27,34–36, 40] allows users to submit queries that not only specify "what" the user is searching for, e.g., in the form of keywords or names of consumer products of interest, but also specify "where" information, e.g., street addresses, postal codes, or geographic coordinates like latitude and longitude. For instance, a search may request a "good micro-brewery that serves pizza" and that is close to the user's hotel. In general, a spatial keyword query retrieves a set of spatial web objects that are located close to the query location and whose text descriptions are relevant to the query keywords. Figure 1(a) illustrates an example spatial keyword query q (located at the dot) with keywords 'outdoor seating' that requests the top-5 restaurants (denoted by squares) in London from TripAdvisor[1] based on a scoring function that performs a weighted sum on the spatial distance and text relevance of restaurants.

Several variants of the spatial keyword query have been studied recently. They differ in terms of the query arguments and in how the relevance of the objects to the query arguments is determined (scoring functions). A continuously moving top-k spatial keyword query [36,37] requests an up-to-date result while the query location changes continuously. A location-aware top-k prestige-based text retrieval query [6] increases the ranking of objects that have highly ranked nearby objects. A collective spatial keyword query [7] retrieves a set of objects that together match query keywords. Rocha-Junior and Nørvåg [28] consider spatial keyword search in road networks, and Li et al. [20] study the spatial keyword search constrained by the moving direction.

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

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

The majority of existing proposals retrieve single objects as elements in the result. However, users may be more interested in regions, or areas, that satisfy their query needs rather than in a set of objects that are scattered in space. For instance, a user who is interested in purchasing jeans may prefer to visit a region with several shops that sell jeans, rather than visit shops that are further apart. In addition, similar objects (with the same functionality) are often located close to each other and thus form small regions, such as shopping, dinning, and entertainment districts [13]. Previous studies [11, 22, 32] consider the co-location relationship between objects and retrieve regions so that the total weights of objects inside the result regions are maximized. However, these proposals impose specific shapes on the result regions, either fixed-size rectangles or circles. A recent study [8] 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. Nevertheless, this kind of query may re-

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

[1]http://www.tripadvisor.com

ACM ISBN .
DOI:

trieve a region containing many objects with low scores and may ignore a promising region with few objects with high scores. Also, the query includes a query range that reduces the search space. The performance is unknown if the query range is set to cover the whole data set.

We aim at a solution that supports browsing, or exploratory, user behavior. It should have no constraints on the shapes of the retrieved regions. To this end, we view regions of interest 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 spatial web 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 (restaurants in London from TripAdvisor) are illustrated in the figure.

Clusters can be found using different approaches. We adopt density-based clustering that has several advantages: (i) no need to specify the number of clusters, (ii) clusters can have arbitrary shapes, and (iii) clusters are robust to outliers. The two basic steps for top-k STC retrieval are (i) obtaining the objects that are relevant to the query keywords and (ii) applying a density based clustering algorithm to these objects to find the top-k clusters. The clustering algorithm processes the objects in a pre-defined order. It terminates when no cluster with a better score can be found. We consider a cluster scoring function (introduced in Section 2) that favors clusters close to the query location and that contain objects with high relevance with regard to the query keywords. Different query keywords result in different clusters. Figure 2 demonstrates the top-5 clusters of restaurants obtained by two queries located at the same location (black dot), but with different keywords, namely 'local cuisine' and 'dessert'. We are not aware of the query keywords until a query arrives, so pre-computing the clusters for all possible sets of query keywords is infeasible. We target an efficient algorithm that is able to find top-k clusters with a response time that supports interactive search.

(a) 'local cuisine' (b) 'dessert'

Figure 2: Example Top-5 Clusters

More specifically, we propose a basic algorithm that combines on-line density-based clustering with an early stop condition. This algorithm applies the state-of-the-art density based clustering algorithm DBSCAN [14] to objects indexed by an IR-tree [34] to find top-k clusters. This algorithm has to check the neighborhood of each relevant object in order to identify dense neighborhoods, since a cluster found by DBSCAN consists of core objects and their dense neighborhoods. We propose an advance algorithm that reduces the number of objects to be examined. Moreover, determining whether a neighborhood is dense or not is time-consuming, since it involves range queries on the IR-tree. We design spatially gridded posting lists (SGPL) to estimate the selectivity of range queries on the IR-tree, so that sparse neighborhoods can be detected quickly without querying the index, thus saving computational cost. Further, SGPL is able to handle the necessary range queries more efficiently compared with the basic algorithm.

To the best of our knowledge, we are the first to study the top-k spatial textual clusters query. This paper's contributions are:

- We introduce the top-k spatial textual clusters query that returns close, relevant, and dense clusters in order to support exploratory user behavior.
- A basic algorithm that exploits the density-based algorithm DBSCAN with an early stop condition on the IR-tree.
- In order to provide response times that enable interactive search, we propose an advanced approach that includes the following techniques.
  - In order 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.
  - A result cluster consists of dense neighborhoods. However, determining whether a neighborhood is dense in the basic algorithm involves an expansive range query on the index. We design spatially gridded posting lists (SGPL) to estimate the selectivity of range queries, thus enabling the pruning of sparse neighborhoods.
  - The SGPL is also able to support range queries, so that a performance gain is obtained for the range queries that cannot be avoided, compared with the basic algorithm.
- An extensive empirical study with real data demonstrates that the paper's proposals offer scalability and are capable of excellent performance.

The rest of the paper is organized as follows. Section 2 formally defines the top-k spatial textual cluster query. The basic algorithm is presented in Section 3. We present the advanced approach in Section 4, including three techniques: (i) a skipping rule that reduces the number of objects to be examined (Section 4.1), (ii) the spatially gridded posting lists (SGPL) for selectivity estimation of range queries (Section 4.2), and (iii) a fast range query processing algorithm on SGPL (Section 4.3). We report on the empirical performance study in Section 5. Finally, we cover related work in Section 6 and offer conclusions and research directions in Section 7.

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 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 [29] or language models [26].

We adopt the density-based clustering model [14], and clusters are query-dependent. We proceed to present the relevant definitions extended for the top-k spatial textual cluster query.

Definition 1. Given a set of keywords ψ, the **relevant object set** D_ψ satisfies (i) $D_\psi \subseteq \mathcal{D}$ and (ii) $\forall p \in D_\psi (\psi \cap p.\psi \neq \emptyset)$.

Definition 2. The **ϵ-neighborhood** of a relevant object $p \in D_\psi$, denoted by $N_\epsilon(p)$, is defined as $N_\epsilon(p) = \{p_i \in D_\psi \mid \|p\ p_i\| \leq \epsilon\}$.

Definition 3. A **dense ϵ-neighborhood** of a relevant object $N_\epsilon(p)$ contains at least $minpts$ objects, i.e., $|N_\epsilon(p)| \geq minpts$.

Definition 4. A relevant object p is a **core** if its ϵ-neighborhood is dense.

Definition 5. A relevant object p_i is **directly reachable** from a relevant object p_j with regard to ϵ and $minpts$ if

1. $p_i \in N_\epsilon(p_j)$ and
2. $|N_\epsilon(p_j)| \geq minpts$.

Definition 6. A relevant object p_i is **reachable** from a relevant object p_j with regard to ϵ 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 $1 \leq m < n$.

Definition 7. A relevant object p_i is **connected** to a relevant object p_j with regard to ϵ 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 ϵ and $minpts$.

Definition 8. A **spatial textual cluster** R with regard to ψ, ϵ, and $minpts$ satisfies the following condition:

1. $R \subseteq D_\psi$ and
2. R is a maximal set such that $\forall p_i, p_j \in R$, p_i and p_j are connected through dense ϵ-neighborhoods when considering only objects in D_ψ.

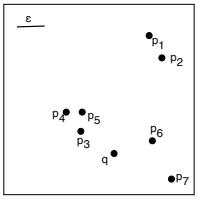
A spatial textual cluster is a density-based cluster [14] found from the relevant object set D_ψ with regard to the query keywords ψ. A **top-k Spatial Textual Cluster** (k-STC) query $q = \langle \lambda, \psi, k, \epsilon, minpts \rangle$ takes five arguments: a point location λ, a set of keywords ψ, a number of requested object sets k, a distance constraint ϵ on neighborhoods, and the minimum number of objects $minpts$ in a dense ϵ-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 ϵ and $minpts$ are able to capture how far the user is willing to move before reaching another place of interest. They depend on the users' preferences.

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 α is used to balance the spatial proximity and the text relevance of the retrieved clusters. Note that all spatial distances and text relevances are normalized to the range $[0, 1]$.

Example 2.1: Consider the example k-STC query q with location $q.\lambda$ and $q.\epsilon$ as shown in Figure 3(a) and with $q.\psi = \{coffee, tea\}$, $q.k = 1$, and $q.minpts = 2$. The data set contains the 7 objects $p_1, p_2, \cdots, p_7$ shown in Figure 3(a). Figure 3(b) shows the document vector and the Euclidean distance to the query location of 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 ($= 0.5 \times (0.11 + 0.15)/2 + (1 - 0.5) \times (1 - (0.5 + 0.5)/2)$). □



(a)

	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

(b)

Figure 3: Example k-STC Query

3. BASIC APPROACH

We first briefly introduce the index structures used for organizing objects and then present the basic algorithm for the processing of k-STC queries.

3.1 Indexes

We adopt the IR-tree [34] and inverted file [41] index structures to organize objects.

An inverted file index has two main components.

- A vocabulary of all distinct words appearing in the text descriptions of the objects in the data set.
- 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 is an R-tree [16] extended 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 spatial 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 the derivation of an upper bound on the text relevance to a query of any object contained in the subtree rooted at e.

Example 3.1: Table 1 shows the inverted file indexing the 7 objects in Figure 3. For example, the posting list for word 'pizza' tells that the text description of p_4 contains 'pizza' and the weight is 0.5. Figure 4 illustrates the IR-tree with fanout 2 indexing the 7 objects in Figure 3. As a specific example, the weight of 'coffee' for entry N_6 in inverted file IF_7 is 0.5, which is the maximal weight of 'coffee' in the two documents in the child node of N_6. □

Table 1: Example Inverted File

coffee	$(p_3, 0.5), (p_6, 0.5), (p_7, 0.5), (p_1, 0.2), (p_2, 0.2)$
tea	$(p_5, 0.5), (p_7, 0.5), (p_1, 0.2), (p_2, 0.2)$
pizza	$(p_4, 0.5)$

3.2 Algorithm

Given a query, the top-k spatial textual clusters are the top-k density-based clusters found from the relevant object set D_ψ with regard to the query keywords. A straightforward solution is to first obtain the relevant object set D_ψ and then to find all density-based clusters in D_ψ. These clusters are then sorted according to the scoring function (Equation 1), and the top-k best clusters are returned as the result. This straightforward solution is inefficient, since finding all clusters is expensive. The proposed basic algorithm is able

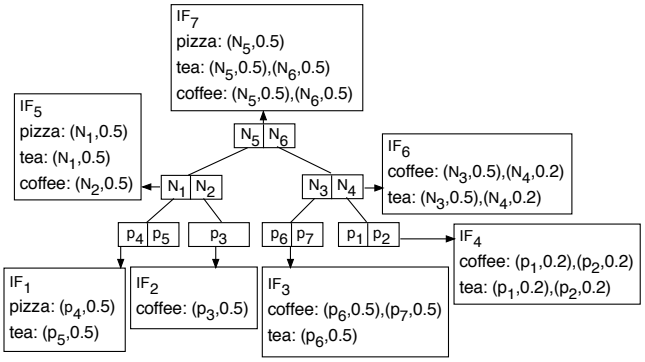


Figure 4: Example IR-tree

to return the top-k clusters without first finding all clusters. Specifically, some candidate clusters are first obtained. 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.

Algorithm 1 shows the pseudo code of the basic algorithm. It first obtains the relevant object set D_ψ 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_ψ in ascending order of their Euclidean distances to the query location, i.e., $d_{q.\lambda}(p.\lambda)$, and keeps the sorted copy in *slist* (line 2). In addition, the objects in D_ψ 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 to each of the two sorted lists *slist* and *tlist* (line 7). Specifically, the algorithm accesses the top member of each of the lists under sorted access, then accesses the second member of each of the lists, and so on. 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 τ 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 finalized as the result.

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

PROOF. The stop condition is satisfied when either of the following two conditions is true: (i) $slist = \emptyset$ or (ii) $bound \geq \tau$. Condition (i) means that all objects have been clustered or identified as noise. Hence, the top-k candidate clusters are the final result, since no further clusters can be found. Consider condition (ii). For any unfound cluster R,

$$\begin{aligned} score_q(R) &= \alpha \cdot d_{q.\lambda}(R) + (1 - \alpha) \cdot (1 - tr_{q.\psi}(R)) \\ &\geq \alpha \cdot sb + (1 - \alpha) \cdot tb \\ &= bound \geq \tau. \end{aligned}$$

Hence, no cluster can have a better (smaller) score than does the top-k candidates. □

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

1: $D_\psi \leftarrow$ **LoadRelevantObjects**$(q.\psi, iindex)$;
2: $slist \leftarrow$ sort objects in D_ψ in ascending order of $d_{q.\lambda}(p.\lambda)$;
3: $tlist \leftarrow$ sort objects in D_ψ in ascending order of $1 - tr_{q.\psi}(p.\psi)$;
4: $rlist \leftarrow \emptyset$;
5: $\tau \leftarrow \infty$;
6: **repeat**
7: Object $p \leftarrow$ sorted access in parallel to $slist$ and $tlist$;
8: $c \leftarrow$ **GetCluster**$(p, q, irtree, tlist, slist)$;
9: **if** $c \neq \emptyset$ **then**
10: Add c to $rlist$;
11: $\tau \leftarrow$ score of the k-th cluster in $rlist$;
12: $sb \leftarrow$ **First**$(slist)$;
13: $tb \leftarrow$ **First**$(tlist)$;
14: $bound \leftarrow \alpha \cdot sb + (1 - \alpha) \cdot tb$;
15: **until** $bound \geq \tau \vee slist = \emptyset$
16: Return $rlist$;

To retrieve a cluster R containing p as a core object, function **GetCluster** (see 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 ϵ-neighborhood of p is dense. If the result set $neighbors$ of the range query contains fewer than $q.minpts$ objects (a sparse neighborhood), 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). Next, the temporary cluster is expanded by checking the ϵ-neighborhood of each object p_i in $neighbors$ except p (lines 12 and 13). If the ϵ-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). In order to avoid duplicate operations, the objects that are marked as noise or are added to the temporary cluster are removed from lists *tlist* and *slist* (lines 4, 9, and 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 ϵ-neighborhood of an object p using an IR-tree on the objects. A priority queue organizes the nodes to be visited in the IR-tree, using the minimum Euclidean distance between nodes and the query as the key. The queue is initialized as empty (line 2). 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 the result $neighbors$ (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.

Example 3.2: Consider the k-STC query in Example 2.1. Given $q.\psi = \{coffee, tea\}$, the basic algorithm first retrieves $D_\psi = \{p_1, p_2, p_3, p_5, p_6, p_7\}$ using the inverted file shown in Table 1. Then, the two sorted lists are calculated, i.e., $slist = ((p_6, 0.1), (p_3, 0.11), (p_5, 0.15), (p_7, 0.19), (p_2, 0.2), (p_1, 0.25))$ and $tlist = ((p_7, 0), (p_3, 0.5), (p_5, 0.5), (p_6, 0.5), (p_1, 0.6), (p_2, 0.6))$. By accessing the two sorted lists in parallel, p_7 is obtained first and is used to obtain a cluster. However, the neighborhood of p_7 is sparse, so p_7 is marked as noise and is removed from lists *slist* and *tlist*.

Algorithm 2 $\mathbf{GetCluster}$(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 $\mathbf{RangeQuery}$(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;
```

Next, p_6 is also marked as noise and is removed from $slist$ and $tlist$. The algorithm proceeds to consider p_3 and obtains a cluster $R = \{p_3, p_5\}$. Then τ is set to the score of R, i.e., 0.315. The two sorted lists become $slist = ((p_2, 0.2), (p_1, 0.25))$ and $tlist = ((p_1, 0.6), (p_2, 0.6))$. The bound is calculated as $bound = 0.5 \times 0.2 + 0.5 \times 0.6 = 0.4$. Thus, we get $bound \geq \tau$. The algorithm can then safely return R as the top-1 result without processing p_1 and p_2. □

4. ADVANCED APPROACH

The basic approach is inefficient due to two main reasons.

- It checks the neighborhoods of all relevant objects with regard to the query keywords, which is expensive.
- Checking the neighborhood of an object involves a time-consuming range query on the index.

The advanced approach includes three techniques that eliminate the above disadvantages and achieve improved performance.

4.1 Object Skipping

Function $\mathbf{GetCluster}$ tries to find a cluster R containing a relevant object p as a core object. It first determines whether the ϵ-neighborhood of p is dense. If so, cluster R is initialized as the set of relevant objects inside the ϵ-neighborhood of p. We consider an object as examined if its neighborhood has been checked. Next, the relevant objects other than p inside the ϵ-neighborhood of p are examined one by one. If a neighborhood under consideration is dense, the newly found relevant objects inside it are added to cluster R. This way, cluster R is finalized when each relevant object has been examined. However, it is possible to get cluster R by examining only a portion of the relevant objects. Consider the example in Figure 5. The neighborhoods of the four objects p_1, p_2, p_3, and p_4 are illustrated by dashed and solid circles. It can be seen that 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). In this case, checking the neighborhood of p_4 is unnecessary after having examined p_1, p_2, and p_3, since the objects inside the neighborhood of p_4 are guaranteed to have already been considered. In other words, examining p_4 cannot contribute more objects to the cluster. 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.

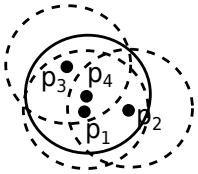


Figure 5: Neighborhoods of Objects

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 $N_\epsilon(p_i) \subset \cup_{1 \leq j < i} N_\epsilon(p_j)$, where $N_\epsilon(p_i)$ is represented as a circular region centered at p_i and with radius ϵ.*

The skipping rule is effective when given a good ordering S. Consider the example in Figure 5. 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. We propose an algorithm **OS** that implements the skipping rule. It follows Algorithm 2 with the following differences.

- Given an object p and its neighborhood, the objects inside the neighborhood are sorted in descending order of their distance to p. The motivation 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 $\mathbf{RangeQuery}$ are sorted in descending order of their distances. Let $S(p)$ be the sorted list of the objects inside the neighborhood of p. Note that 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.
- 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 implementation of 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 [36].

4.2 Spatially Gridded Posting Lists

In the previous section, we proposed a technique that reduces the number of objects to be exmained. However, for those objects that cannot be skipped, checking their neighborhoods involves time-consuming range queries. The result of checking a neighborhood is that the neighborhood is either dense or sparse. In this section, we 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 6(a) illustrates a 4×4 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 6(c) shows the SGPLs for words 'coffee' and 'tea'. For example, the first entry in the SGPL for 'coffee' tells that the document of p_3 contains 'coffee' and 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 range query. We define a merging operator $\bigoplus$ on several 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.,

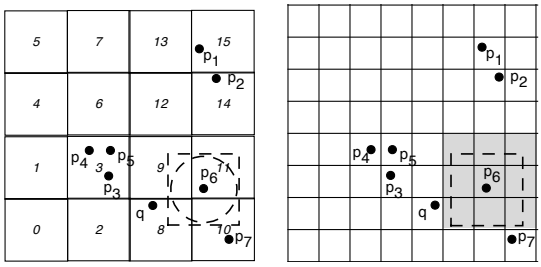
$$\bigoplus_{w_i \in q.\psi} SGPL_{w_i} = \{(c_j, |\bigcup_{w_i \in q.\psi} S_{w_i,c_j}|\}, \quad (2)$$

where $q.\psi$ is the query keywords.

Example 4.2: Consider the example SGPLs in Figure 6(c). The third row 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. □

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 ϵ (e.g., the dashed circle in Figure 6(a)). We approximate the circle q_c as its circumscribed square q_s (e.g., the dashed square in Figure 6(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. Note that it is not necessary to merge the whole SGPL of each query keyword. For the sake of efficiency, only the cells that intersect q_s need to be considered. We thus define a parameterized merging operator $\bigoplus(q_s)$ as follows.

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



(a) 4×4 Grid (b) 8×8 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 6: Example Spatially Gridded Posting Lists

Consider the example in Figure 6 where q_s is the dashed square. We have $coffee \bigoplus(q_s) tea = \{(10, 1), (11, 1)\}$. Based on the coding scheme of the space-filling curve, we adopt an efficient existing algorithm [19] 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 ϵ-neighborhood of object p is guaranteed to be sparse. Otherwise, function $\mathbf{RangeQuery}$ is called to compute the exact number of objects in the ϵ-neighborhood of object p. Hence, some sparse ϵ-neighborhoods can be found efficiently. However, it is not guaranteed that all sparse relevant ϵ-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: Suppose $q.minpts = 2$, meaning that a dense ϵ-neighborhood should contain at least 2 objects. In Figure 6(a), the dashed square q_s approximates the ϵ-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 ϵ-neighborhoods of p_6 is conservatively assessed as not sparse, and function $\mathbf{RangeQuery}$ has to be called to compute the exact number of objects in the ϵ-neighborhood. However, if using the finer grid in Figure 6(b), the selectivity is 1 ($< q.minpts$). Only the gray cells intersect the query range. Thus, the ϵ-neighborhood of p_6 is guaranteed to be sparse, and there is no need to call function $\mathbf{RangeQuery}$. □

We observe that using a finer grid may improve the quality of the selectivity estimation so that expensive $\mathbf{RangeQuery}$ operations are avoided. However, the cost of the selectivity estimation increases when using a finer grid. In the empirical study, we study how the performance is affected by the granularity of the grid.

4.3 FastRange

In addition to supporting selectivity estimation, SGPLs are able to support the processing of range queries that are issued on the IR-tree. We propose an algorithm $\mathbf{FastRange}$ that handles range queries on SGPLs. Before presenting the algorithm, we first override the parameterised merging operator $\bigoplus(q_s)$ as $\overline{\bigoplus}(q_s)$.

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

The result of operator $\bigoplus(q_s)$ records the *number* of objects inside

each cell intersecting query range q_s, while the result of operator $\overline{\bigoplus}(q_s)$ contains the *set of the identifiers* of the objects inside each cell intersecting query range q_s. Algorithm 4 shows the pseudo code of $\mathbf{FastRange}$ that takes two arguments: *list* is the result of operator $\overline{\bigoplus}(q_s)$, and q_c is a circular region centered at an object p and with radius ϵ. If a cell c from *list* is completely inside 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 ϵ 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

We conduct empirical studies to evaluate our proposals. Section 5.1 presents the data set, queries, parameters, and platform used in the experiments. The proposals for k-STC queries are evaluated in Section 5.2.

5.1 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 using TripAdvisor in the following way, such that the distribution of the real data set is roughly maintained. 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.

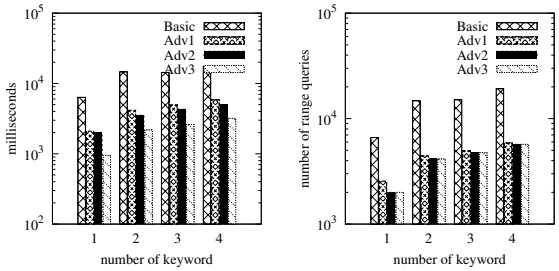
We generate 4 query sets in the space of the dataset, in which the number of keywords is 1, 2, 3, and 4, respectively. Each set comprises 100 queries. Queries are generated from objects, and we guarantee that no query has an empty result. Specifically, to generate a query, we randomly pick an object in the dataset and take the location of the object as the query location and randomly choose words from the document of 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 2 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. We report the average elapsed time cost and the average number of range queries issued needed to compute the k-STC queries.

5.2 Performance Evaluation

We evaluate our proposals against the basic algorithm under varying parameter settings.

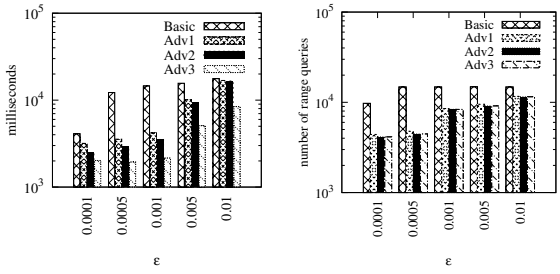
Varying the Number of Keywords $|q.\psi|$. Figure 7 shows the performance of the four approaches when varying the number of query keywords. Their performance becomes worse as the number of query keywords increases, since more objects are involved so that the search space increases. The computation costs of the advanced algorithms increases more slowly than that of the basic algorithm. As expected, the number of range queries issued (Figure 7(b)) is consistent with the elapsed time (Figure 7(a)).



(a) Elapsed time (b) # of range queries

Figure 7: Varying the Number of Keywords

Varying Density Requirement. The density requirement involves two parameters, ϵ and $minpts$. Figure 8 shows the performance of the four approaches when varying ϵ. Figure 9 shows their performance when varying $minpts$. The y axes are logarithmic. A large ϵ or a small $minpts$ indicate a low density requirement. As ϵ increases, the performance becomes worse. The reason is that a low density requirement renders more objects' neighborhoods dense. Recall the basic idea of our proposals. If the ϵ-neighborhood of a relevant object satisfies the density requirement, we need to examine the relevant objects inside the ϵ-neighborhood to expand the cluster. Hence, more dense neighborhoods incur additional computation cost. However, the performance is not sensitive to $minpts$ in range $[10, 200]$ given that $\epsilon = 0.001$. To altering the density requirement, we may choose to either vary ϵ or vary $minpts$. It is of no interest in finding the value range of $minpts$ that affects the performance for the interest of space.



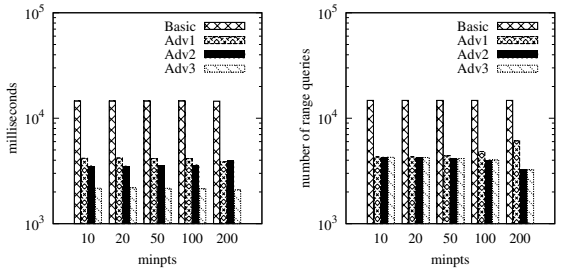
(a) Elapsed time (b) # of range queries

Figure 8: Varying ϵ

Varying α. Figure 10 shows the elapsed time and the number of range queries of the four approaches when varying α that balances the weight between the spatial distance and the text relevance in the scoring function. Their performance is insensitive to parameter α. Hence, our approach can guarantee a good performance no matter what users' preferences are.

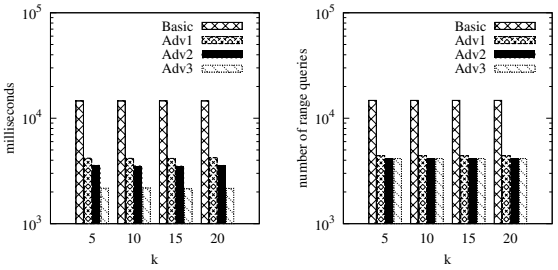
Table 2: Parameter Values

Interpretation	Parameter	Values
Number of requested clusters	k	5, **10**, 15, 20
Number of query keywords	$\|q.\psi\|$	1, **2**, 3, 4
Density requirements	ϵ	0.0001, 0.0005, **0.001**, 0.005, 0.01
	$minpts$	10, 20, **50**, 100, 200
Order of Z-curve in SGPL	h	3, 4, 5, **6**, 7, 8, 9, 10
Balancing the weights between spatial distance and text relevance	α	0.1, 0.3, **0.5**, 0.7, 0.9
Data set size	$\|\mathcal{D}\|$	**100K**, 200K, 400K, 600K, 800K, 1M



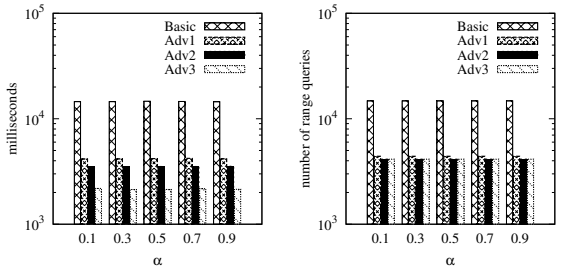
(a) Elapsed time (b) # of range queries

Figure 9: Varying $minpts$



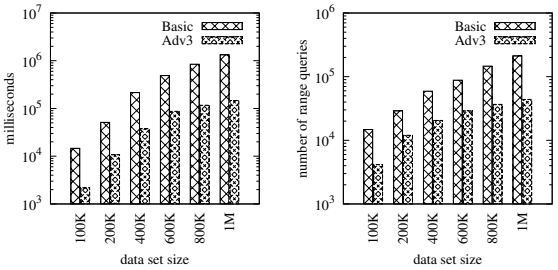
(a) Elapsed time (b) # of range queries

Figure 11: Varying k



(a) Elapsed time (b) # of range queries

Figure 10: Varying α



(a) Elapsed time (b) # of range queries

Figure 12: Varying Data Size

Varying the Number k of Requested Clusters. Figure 11 shows the elapsed time and the number of range queries of the four approaches when varying the number k of requested clusters. Their performance is insensitive to parameter k. For a small k, the reason may be that the threshold in the basic algorithm is not tight enough to prune the clusters that are beyond the top-k result early. For a large k, the reason may be that the total number of clusters found from the whole data set cannot exceed k.

Scalability. Figure 12 shows how the performance of the basic algorithm and the advanced algorithm change as the size of the data set increases. We observe that both algorithms scale linearly. However, the computation cost of the advanced algorithm increases more slowly that does that of the basic algorithm.

Varying the Order of Z-Curve h in SGPL. SGPL is constructed based on grid system over the data set, indexed by a space filling curve. We adopt the Z-curve in our evaluation. Other types of space filling curves are also applicable. The order h of the Z-curve defines the granularity of the grid system that affects the performance of SGPL. A large h provides a finer granularity, so that the ability to estimate the selectivity of range queries and the ability to detect sparse ϵ-neighborhoods are improved. The evaluation result in Figure 13 makes this point. As h increases, the performance improves. We observe that when h exceeds 6, the performance of SGPL becomes stable. This means that the selectivity estimation ability is close to optimal. It is not necessary to devote efforts to construct an SGPL using an even finer grid.

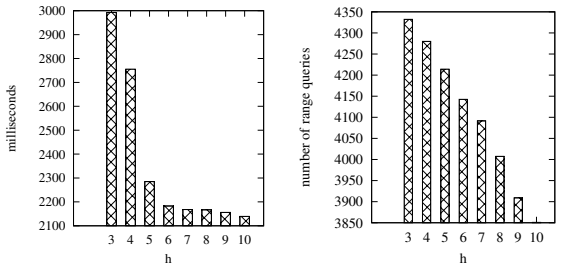
Summary. Overall, the best advanced approach outperforms the basic approach by an order of magnitude. The elapsed time and the number of range queries are proportional to each other, which indicates that the time consuming part of the k-STC queries is the range queries. The proposed advanced approach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

6. RELATED WORK

The problem studied in the paper is related to two topics: spatial textual search, surveyed in Section 6.1, and density-based clustering, surveyed in Section 6.2.

6.1 Spatial Textual Search

Spatial keyword search has attracted much attention in recent



(a) Elapsed time (b) # of range queries

Figure 13: Varying the Order h in SGPL

years. A spatial keyword query retrieves spatial web objects (e.g., web content about shops and restaurants) that are spatially and textually relevant to query arguments. Several efficient geo-textual indexes have been proposed to support spatial keyword queries. Most indexes combine the R-tree for spatial indexing and inverted files/signature files for text indexing, such as the IR^2-tree [15], the IR-tree [34], and the S2I [27]. These hybrid index structures are able to utilize both the spatial and textual information to prune the search space when processing a query. Our proposal is orthogonal to the above and similar indexes. We adopt the IR-tree [34] in the basic algorithm. Other R-tree based indexes can also be used, and the resulting performance can be expected to be comparable to that of the basic algorithm.

Next, a range of studies investigate interesting variants of the prototypical spatial keyword query that satisfy different users' needs. Chen et al. [9] study the problem of matching a stream of incoming Boolean range continuous queries against a stream of incoming geo-textual objects in real time. The keyword-aware optimal route query [4] finds an optimal route that covers a set of user-specified keywords, meets a specified budget constraint, and achieves the best score according to a specific scoring function. The collective spatial keyword query [5, 7, 24] finds a group of objects that are close to a query point and that collectively cover a set of query keywords. Wu et al. [36, 37] cover the problem of maintaining the result set of top-k spatial keyword queries while the user (query location) moves continuously. The Location-aware top-k Prestige-based Text retrieval (LkPT) query [6] retrieves the top-k spatial web objects ranked according to both prestige-based relevance and location proximity, where the prestige-based relevance captures both the textual relevance of an object to a query and the effects of nearby objects. The Reverse Spatial Textual k Nearest Neighbor (RSTkNN) query [25] finds objects that take the query object as one of their k most spatial-textual similar objects.

Recent studies [2, 31] target the common case where the user wishes to find nearby groups of points of interest that are relevant to the query keywords. Such groups are relevant to users who wish to conveniently explore several options before making a decision such as to purchase a particular product. The spatio-textual similarity join [3] retrieves the pairs of objects that are spatially close and textually similar. Zhang et al. [38, 39] study the query that takes a set of keywords and aims to find a set of geo-textual objects such that the union of their text descriptions cover all query keywords and such that the diameter of the objects is minimized. Considering the co-location relationship between objects, regions are retrieved so that the total weights of objects inside the result regions are maximized [11, 22, 32]. However, these proposals impose specific shapes on the result regions, either fixed-sized rectangles or circles. A recent study [8] 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. Also, the query includes a query range that reduce the search space. The performance is unknown if the query range is set to cover the whole data set.

It is well-known that clustering techniques are too expensive for interactive use. However, it is natural to try to leverage the work on clustering for the functionality that the paper targets. This paper is the first to explore the idea of performing clustering at query time. It thus explores a new direction in spatial keyword querying that represents an alternative to existing approaches. Our empirically study also demonstrates that it is indeed possible to adapt clustering to the paper's interactive setting.

6.2 Density-based Clustering Algorithms

In density-based clustering, clusters are defined as regions with a higher density of objects than in the remainder of the space covered by the data set. Objects in the sparse regions that separate clusters are usually considered to be noise and border objects. The most popular density based clustering method is DBSCAN [14]. It features a well-defined cluster model called "density-reachability." It is based on connecting objects within a certain distance threshold of each other. A cluster consists of all density-connected objects together with all objects that are within the threshold distance of these objects, and a cluster can have an arbitrary shape. OPTICS [1] is a generalization of DBSCAN that removes the need to choose an appropriate value for the range parameter and that produces a hierarchical result related to that of linkage clustering. DBRS [33] improves DBSCAN by repeatedly picking an unclassified point at random and examining its neighborhood. VDBSCAN [23] aims for varied-density dataset analysis. Before adopting the traditional DBSCAN algorithm, some methods are used to select several values of parameter ϵ for different densities according to a k-dist plot. With different values of ϵ, it is possible to find clusters with varied densities simultaneously. GDBSCAN [30] clusters point objects as well as spatially extended objects according to both, their spatial and their nonspatial attributes.

The techniques proposed in this paper are presented on the base of DBSCAN. However, it is possible to apply our approach to other density-based clustering models, since DBSCAN serves as the foundation for different variants of density-based clustering.

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 that relate to the application of clustering to spatial-keyword query-

ing. The basic algorithm contains an early stop condition that tries to obtain the top-k clusters without finding all clusters from the whole data set. However, the tightness of the threshold in this early stop condition can be improved. It is of interest to derive a tighter threshold to save computation cost. The density requirements ϵ and $minpts$ in the query define the form of the clusters to be retrieved. However, users may not stick to a fixed density requirement, and the data set may have varying density, e.g., city center vs. countryside. A density requirement good for a city center may miss some results in the countryside. It is of interest to propose an algorithm that is able to automatically select suitable parameters for data sets with varying densities.

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

[4] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.

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

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

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

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

[9] L. Chen, G. Cong, and X. Cao. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*, pages 749–760, 2013.

[10] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pages 277–288, 2006.

[11] 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.

[12] 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.

[13] S. N. Durlauf and L. E. Blume. *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, 2nd edition, 2008.

[14] 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.

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

[16] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[17] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSDBM*, page 16, 2007.

[18] A. Khodaei, C. Shahabi, and C. Li. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *DEXA*, pages 450–466, 2010.

[19] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. In *BNCOD*, pages 20–35, 2000.

[20] G. Li, J. Feng, and J. Xu. Desks: direction-aware spatial keyword search. In *ICDE*, 2012.

[21] 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 Trans. Knowl. Data Eng.*, 23(4):585–599, 2011.

[22] J. Liu, G. Yu, and H. Sun. Subject-oriented top-k hot region queries in spatial dataset. In *CIKM*, pages 2409–2412, 2011.

[23] 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.

[24] 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.

[25] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

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

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

[28] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, 2012.

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

[30] 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.

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

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

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

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

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

[36] D. Wu, M. L. Yiu, and C. S. Jensen. Moving spatial keyword queries: Formulation, methods, and analysis. *ACM Trans. Database Syst.*, 38(1):7, 2013.

[37] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

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

[39] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532, 2010.

[40] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005.

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