Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

Density-Based Top-K Spatial Textual Clusters Retrieval

Wu, Dingming; Keles, Ilkcan; Wu, Song; Zhou, Hao; Saltenis, Simonas; Jensen, Christian S.; Lu, Kezhong

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2021.3049785

Publication date:
2022

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Wu, D., Keles, I., Wu, S., Zhou, H., Saltenis, S., Jensen, C. S., & Lu, K. (2022). Density-Based Top-K Spatial Textual Clusters Retrieval. *IEEE Transactions on Knowledge and Data Engineering*, *34*(11), 5263-5277. https://doi.org/10.1109/TKDE.2021.3049785

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: May 04, 2024

Density-Based Top-K Spatial Textual Clusters Retrieval

Dingming Wu, Ilkcan Keles#, Song Wu#, Hao Zhou, Simonas Šaltenis, Christian S. Jensen, and Kezhong Lu✉

Abstract—So-called spatial web queries retrieve web content representing points of interest, such that the points of interest have descriptions that are relevant to query keywords and are located close to a 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, the top-k spatial textual cluster retrieval (k-STC) query that returns the top-k clusters that (i) are located close to a query location, (ii) contain objects that are relevant with regard to given query keywords, and (iii) have an object density that exceeds a given threshold. To compute this query, we propose a DBSCAN-based approach and an OPTICS-based approach that rely on on-line density-based clustering and that exploit early stop conditions. Empirical studies on real data sets offer evidence that the paper's proposals can find good quality clusters and are capable of excellent performance.

1 Introduction

Spatial keyword query processing [1]–[12] allows users to receive answers to geographically constrained queries that take into account information about "what" the user is searching for as expressed by keywords. For instance, a query may request a "good micro-brewery that serves pizza" 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 a query q (black dot) with keywords 'outdoor seating' that requests the top-5 restaurants (red squares) in London from TripAdvisor[1] based on a ranking function that is a weighted sum of spatial distance and text relevance. Several spatial keyword query variants have been studied. Proposals differ in terms of the query arguments and in how the objects matching the query arguments are found and ranked. A continuously moving top-k spatial keyword query [9], [13] requests the up-to-date result while the query location changes continuously. A location-aware top-k prestige-based text retrieval query [14] gives high rankings to the objects that have relevant surrounding objects. A collective spatial keyword query [15] retrieves a set of objects that taken together best match the query arguments. Spatial keyword queries are also investigated in road networks [16], [17], and Li et al. [18] study spatial keyword search constrained by a movement direction.

Most studies consider the retrieval of one or more objects, each of which satisfies the query. However, in some use cases, users may be interested in regions with many objects that satisfy query parameters rather than in a set of objects scattered in space. For instance, a user may prefer to visit one nearby shopping area to explore multiple outlets selling jeans, rather than visiting one jeans shop in zone A and another in zone B. Such functionality is also useful for a marketing manager, who wishes to get an overview of the locations of coffee shops in a central business district. In addition, similar businesses are often located close to each other and form small regions, such as shopping, dinning, and entertainment areas, to attract customers [19]. Some previous studies [20]–[22] consider co-location relationship between objects and retrieve regions so that the total weights of objects inside the regions are maximized. However, these studies are limited regarding the shapes of the regions retrieved, such as a fixed-size rectangle or a circle. A recent study [23] requests the length-constrained maximum-sum region of interest where the road network distance between objects is less than a query constraint and the sum of the ranking scores of the objects inside the region is maximized. This kind of query may still retrieve a region containing many objects with low ranking scores and may ignore promising regions with few objects with high ranking

- *D. Wu, K. Lu, S. Wu, and H. Zhou are with the College of Computer Science and Software Engineering, Shenzhen University, China. E-mail: {dingming, kzlu}@szu.edu.cn, {wusong2018, zhouhao2017}@email.szu.edu.cn*
- *I. Keles is with Turkcell, Turkey. E-mail: ilkcan.keles@turkcell.com.tr*
- *I. Keles, S. Saltenis, and C. S. Jensen are with the Department of Computer Science, Aalborg University, Denmark. E-mail: {ilkcan, simas, csj}@cs.aau.dk*
- *#: These authors contributed equally to this study and share second authorship.*

1. http://www.tripadvisor.com

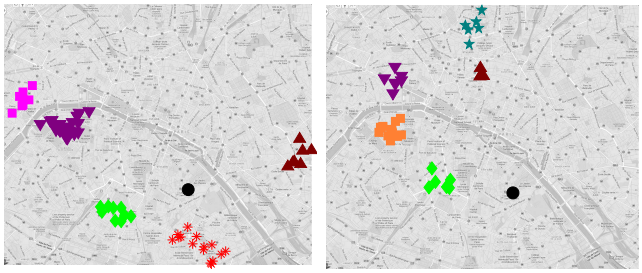
scores. Also a query range must be specified, which helps reduce the search space.

We aim at a solution supporting convenient exploration of nearby web objects. It should have no constraints on the shapes of retrieved regions. To this end, we consider the regions of interest as spatial textual clusters 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 regard to 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 shown in Figure 1(b).

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

Fig. 1. Top-k Objects vs. Top-k Clusters

Finding clusters can be achieved using different approaches. We adopt density-based clustering due to the following advantages: (i) no need to specify the number of clusters, (ii) clusters may have arbitrary shapes, and (iii) clusters are robust to outliers. The basic steps for the 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 find the top-k clusters of the relevant objects. The clustering algorithm processes the objects in a predefined order. It terminates when no cluster with a better ranking score can be found. We consider a function to rank clusters (introduced in Section 2) that favors clusters close to the query location that contain objects with high textual relevance to the query keywords. Different query keywords result in different clusters. For example, Figure 2 shows the top-5 clusters of two queries with the same location (black dot) but keywords 'local cuisine' versus 'dessert' when applied to the restaurants in London from TripAdvisor. The query keywords are unknown until a query arrives. Pre-computing the clusters for all possible query keywords is computationally prohibitive. We thus target efficient solutions that are able to compute on the fly top-k clusters with good response times.



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

Fig. 2. Example Top-5 Clusters

Based on the density-based clustering model, we study DBSCAN and OPTICS based approaches to the efficient processing of k-STC queries. The DBSCAN-based approach targets scenarios where users have explicit density requirements, while the OPTICS-based approach targets the cases where density parameters are difficult to specify.

The DBSCAN-based approach applies the state-of-the-art density based clustering algorithm DBSCAN [24] together with the IR-tree [10] to find the top-k clusters. The original DBSCAN algorithm has to check the neighborhood of each relevant object in order to identify dense neighborhoods because a cluster found by DBSCAN consists of several core objects and their dense neighborhoods. We improve this by reducing the number of objects to be examined. Moreover, determining whether a neighborhood is dense or not is time-consuming, since it involves issuing range queries. We design spatially gridded posting lists (SGPL) to estimate the selectivity of range queries so that sparse neighborhoods can be detected quickly, thus saving computational cost. The SGPL also enable processing of the remaining necessary range queries more efficiently compared to using the IR-tree.

The OPTICS-based approach extends the state-of-the-art algorithm OPTICS [25] and generates an ordering of relevant objects w.r.t. the query keywords, which represents the inherent density-based clustering structure of the data. The top-k clusters are extracted from this structure. We observe that simply using the original OPTICS algorithm is inefficient. Instead, we propose an algorithm OPTICS-OM based on a word-ordering index that avoids computing clusters from scratch, reducing the computational cost of the k-STC query significantly.

The rest of the paper is organized as follows. Section 2 formally defines the top-k spatial textual cluster query. The DBSCAN-based approach is proposed in Section 3. The OPTICS-based approach is presented in Section 4. We report on an empirical performance study in Section 5. Finally, we cover related work in Section 6 and offer conclusions in Section 7.

2 Problem Definition

We consider a data set $\mathcal{D}$ in which each object p 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 different ways, e.g., using tf-idf weighting [26].

We adopt density-based clustering [24], meaning that clusters are query dependent. We proceed to adapt existing definitions to the top-k spatial textual cluster query studied in this paper.

Definition 1: Given a set of keywords ψ, the **relevant object set** D_ψ satisfies (i) $D_\psi \subseteq \mathcal{D}$ and (ii) $\forall p \in D_\psi (tr_\psi(p.\psi) > \theta \wedge p.\psi$ contains at least one keyword in $\psi)$.

Function $tr_\psi(p.\psi)$ is used for evaluating the text relevance of an object p w.r.t. keywords ψ, e.g., using language models [27]. In general, the larger the function value is, the more relevant the object is. System parameter θ is pre-defined to exclude objects with low relevance.

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

Definition 3: An ϵ-neighborhood of a relevant object $N_\epsilon(p)$ is **dense** if it 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) $p_i \in N_\epsilon(p_j)$
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 $1 \leq m \leq n$.

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 conditions:

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

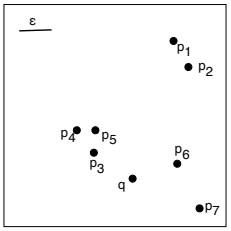
A spatial textual cluster is a density-based cluster [24] found from the relevant object set D_ψ that is parameterized by the query keywords ψ. A **top-k Spatial Textual Cluster** (k-STC) query $q = \langle \lambda, \psi, k, \epsilon, minpts \rangle$ takes five parameters: a point location λ, a set of keywords ψ, a number of requested object sets k, a distance constraint ϵ for 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 indicates that the top-k clusters do not overlap. The density parameters ϵ and $minpts$ indicate intuitively how far a user is willing to travel before reaching another place of interest. They depend on users' preferences.

The k-STC query favors clusters with high text relevance to the query keywords and that are located close to the query location. We use the following scoring function.

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

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 value of the objects in R. Our approaches are not limited to the above scoring function but rather are applicable to any scoring function that is monotone in both the spatial distance and the text relevance. Parameter α is used to balance between the spatial proximity and the text relevance of the retrieved clusters. All spatial distance and text relevance values are normalized to range $[0, 1]$.

Example 2.1: Consider an 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 vectors and the Euclidean distances to the query location of the objects. Let $\alpha = 0.5$ and $tr_{q.\psi}(p.\psi) = \sum_{t \in q.\psi} w_t$, where w_t is the value of the corresponding dimension in $p.\psi$ for term t in the query keywords. The top-1 cluster is $R = \{p_3, p_5\}$ with score 0.155 ($= 0.5 \times 0.11 + 0.5 \times (1 - 0.8)$). □



(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,	0.2,	0)	0.11
p_4	(0,	0,	0.5)	0.18
p_5	(0.3,	0.5,	0)	0.15
p_6	(0.5,	0,	0)	0.1
p_7	(0.5,	0.5,	0)	0.19

(b)

Fig. 3. Example k-STC Query

3 DBSCAN-based Approach

3.1 Basic Algorithm

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

3.1.1 Indexes

We adopt the IR-tree [10] and inverted file [28] index 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 an identifier of an object whose text description contains t and w is the weight of t.

The IR-tree is an R-tree [29] extended with inverted files. Each leaf node contains entries of the form $e_0 = (id, \Lambda)$, where $e_0.id$ refers to an object 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 that indexes the text of all objects stored in the node. Each non-leaf node N in the IR-tree 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 that indexes the pseudo text 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 of 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' that has weight 0.5. Figure 4 illustrates the IR-tree with fanout 2 indexing the same objects. For 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	$(p_3, 0.5), (p_6, 0.5), (p_7, 0.5), (p_5, 0.3), (p_1, 0.2), (p_2, 0.2)$
tea	$(p_5, 0.5), (p_7, 0.5), (p_1, 0.2), (p_2, 0.2), (p_3, 0.2)$
pizza	$(p_4, 0.5)$

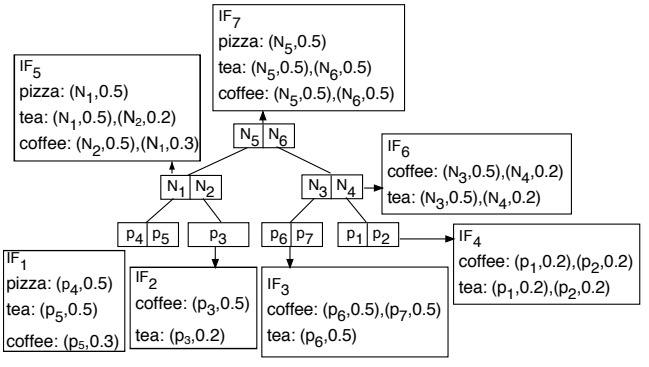


Fig. 4. Example IR-tree

3.1.2 Algorithm

A k-STC query returns the top-k density-based clusters found in the relevant object set D_ψ with regard to the query keywords. A straightforward solution is first obtaining the relevant object set D_ψ and then find all density-based clusters in D_ψ. These clusters are sorted according to the scoring function (Equation 1), and the top-k clusters are returned. This solution is inefficient because finding all clusters is expensive. The complexity of this straightforward solution is $O(N \log N)$, which equals the complexity of DBSCAN. The proposed basic algorithm is able to return the top-k clusters without first computing all clusters. Specifically, candidate clusters are obtained first. A threshold is set as the score of the k-th candidate cluster. The basic algorithm estimates a lower bound on the scores of all unfound clusters. If the bound is worse than the threshold, the top-k candidate clusters constitute the final 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 unioning the posting lists of the query keywords 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 stores the result in $slist$ (line 2). Then, the objects in D_ψ are sorted in descending order of their text relevance to the query keywords, i.e., $tr_{q.\psi}(p.\psi)$, and the result is stored in $tlist$ (line 3). The candidate list $rlist$ is initialized as empty and the threshold is set to infinity (line 4). The algorithm then traverses the two sorted lists in parallel (line 6). Specifically, the algorithm alternates between removing top elements from the two lists. As object p is obtained from one of the lists, function **GetCluster** (Algorithm 2) tries to build a cluster containing p as a core object (line 7). Meanwhile, the objects contained in the cluster are removed from both $slist$ and $tlist$. If the 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 8–10). As detailed later, the algorithm then computes a lower bound $bound$ on the scores of all unfound clusters (line 11). The result is guaranteed to be found when $bound \geq \tau$ or $slist$ is exhausted (indicating that $tlist$ is also exhausted) (line 12). The top-k candidate clusters in $rlist$ are the result.

Before describing how the lower bound of the scores of all unfound clusters is computed, we first consider how clusters are built. 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 ϵ-neighborhood of p is dense. If the result $neighbors$ 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 in the neighborhood that were previously labeled as noise are added to the temporary cluster (lines 16 and 17). Further, the objects in the neighborhood that do not belong to R are added to both the temporary cluster and *neighbors* in preparation for further expansion in the following iterations (lines 18–21). To avoid duplicate operations, the objects that are either marked as noise or added to the temporary cluster are removed from lists *tlist* and *slist* (lines 4, 9, and 20). Temporary cluster R is finalized and returned if no more object can be added.

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

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

Function **RangeQuery** on the IR-tree is used to find the objects that are relevant to the query keywords and located within distance $q.\epsilon$ of p_i, which is the ϵ-neighborhood of object p_i. Details on **RangeQuery** can be found elsewhere [30].

The lower bound of the score is calculated as $bound = \alpha \cdot sb(slist) + (1-\alpha) \cdot (1 - tb(tlist))$ (lines 12), where $sb(slist)$ is a lower bound on the spatial distance (Lemma 1) between the query location and all unfound clusters and $tb(tlist)$ is an upper bound on the text relevance (Lemma 2) of all unfound clusters. To derive these bounds, let D_n contain the objects that are currently marked as noise. Each object $p_n \in D_n$ is associated with a subset N_{p_n} of its ϵ-neighborhood $N_\epsilon(p_n)$, in which the objects have not been processed. As the algorithm proceeds, the objects in N_{p_n} are removed as they are either added to clusters or marked as noise. An object in D_n is removed when it is either added to a cluster or its ϵ-neighborhood is empty. In other words, D_n contains the objects that are temporarily marked as noise, but may be added to clusters later.

Lemma 1: Let s_0 be the spatial distance of the current first object in *slist*. A lower bound of the spatial distance between the query location and all unfound clusters *sb* is $\min(s_0, \min\{d_{q.\lambda}(p_n.\lambda) | p_n \in D_n\})$.

Lemma 2: Let t_0 be the text relevance of the current first object in *tlist*. An upper bound of the text relevance of all unfound clusters *tb* is $\max(t_0, \max\{tr_{q.\psi}(p_n.\psi) | p_n \in D_n\})$.

The proofs of the lemmas are straightforward and thus omitted.

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

1: $R \leftarrow \emptyset$;
2: *neighbors* $\leftarrow$ *irtree*.**RangeQuery**(q, p);
3: **if** $neighbors.size < q.minpts$ **then**
4: Remove p from *tlist* and *slist*;
5: Mark p as 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 $p_i \leftarrow$ remove an object from *neighbors*;
13: $neighbors_i \leftarrow$ *irtree*.**RangeQuery**(q, p_i);
14: **if** $neighbors_i.size \geq q.minpts$ **then**
15: **for** each object p_j in $neighbors_i$ **do**
16: **if** p_j is noise **then**
17: Add p_j to R;
18: **else if** $p_j \notin R$ **then**
19: Add p_j to R;
20: Remove p_j from *tlist* and *slist*;
21: Add p_j to *neighbors*;
22: Return R;

3.2 Enhancements

The basic algorithm is inefficient for two main reasons.

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

Next , we address these inefficiencies.

3.2.1 Object Skipping

Function **GetCluster** tries to find a cluster R containing a relevant object p as a core object. It first determines whether the ϵ-neighborhood of p is dense. If dense, cluster R is initialized as the set of relevant objects inside the ϵ-neighborhood of p. Next, the relevant objects other than p inside the ϵ-neighborhood

of p are examined in turn. An object is examined if its neighborhood has been checked. If a neighborhood under consideration is dense, the newly found relevant objects in it are added to cluster R. This way, cluster R is finalized after every 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). Having already examined p_1, p_2, and p_3, checking the neighborhood of p_4 is unnecessary because all objects in the neighborhood of p_4 have been considered. Based on this observation, we define a skipping rule that reduces the number of relevant objects to be examined and design an algorithm **OS** that implements the rule.

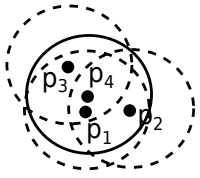


Fig. 5. Neighborhoods of Objects

Skipping Rule: Let $S = (p_1, p_2, \cdots, p_n)$ be an examining order of a set of objects. 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., $N_\epsilon(p_i) \subset \cup_{1 \leq j < i} N_\epsilon(p_j)$, where $N_\epsilon(p_i)$ is a circle centered at p_i with radius ϵ.

The effectiveness of the skipping rule depends on the ordering S. 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 a subsequent object is high. Algorithm **OS** that implements the skipping rule follows Algorithm 2 with the following differences.

- Given an object p and its neighborhood, the objects in 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 the neighborhoods are. Referring to lines 2 and 13 in Algorithm 2, the objects returned from the **RangeQuery** are sorted in descending order of their distances.
- Each time before checking the neighborhood of an object (line 13 in Algorithm 2), the skipping rule is applied. If it is successful, the algorithm continues to process the next object. The implementation of the skipping rule involves the judgement of whether a circle is covered by the union of several circles. This can be done using a recursive subdivision of the circle by non-overlapping squares [9].

3.2.2 Spatially Gridded Posting Lists

Object skipping reduces the number of objects to be exmained. However, for those objects that cannot be skipped, checking involves time-consuming range queries. The result of checking a neighborhood is that it is either dense or sparse. Here, we design **spatially gridded posting lists** (SGPL) to facilitate efficient estimation of the selectivity of a range query on the IR-tree, such that sparse neighborhoods can be determined quickly 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 contains w. Let D_{w_i} be the set of objects containing word w_i. The SGPL of word w_i is a sorted list of entries (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 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 or Z-order curve. The SGPLs of all distinct words in the data set are organized similarly to the inverted file. Note that empty cells are not stored. Given a word, its SGPL can be retrieved.

Let N_w be the number of distinct words, and let N be the number of objects in the data set. On average, the number of objects in a grid cell is N/n^2. The space complexity of the SGPLs is $O(N_w \cdot N \cdot n)$ on average, where n is small compared to N_w and N.

Example 3.2: 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 values of 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 that p_3 is located in cell 3. □

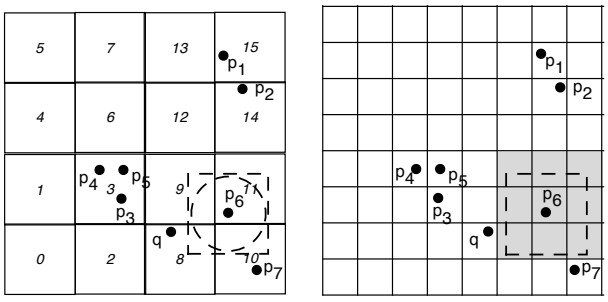
Given a set of m query keywords, the corresponding m SGPLs are merged to estimate the selectivities of range queries. We define a merging operator $\bigoplus$ on several SGPLs that produces a count for each non-empty cell. The count for cell C is the cardinality of the union of the sets of objects located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

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

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

Example 3.3: 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 first entry $(3, 2)$ tells that cell 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



(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

Fig. 6. Example Spatially Gridded Posting Lists

range q_c query centered at an object p with radius ϵ (e.g., the dashed circle in Figure 6(a)). We approximate the circle q_c by 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 square q_s need to be considered. We define a parameterized merging operator $\bigoplus(q_s)$ as follows.

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

The computational cost of the merging operator is proportional to the number of objects in the grid cells intersecting q_s and whose documents contain the query keywords. The size of q_s is small, and the number of objects involved is low in real data sets. Thus, the computational cost of the merging operator is generally low.

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 [31] to retrieve the cells that intersect the query range q_s.

The derived selectivity serves as an upper bound on the number of objects in circle q_c. It 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. If the selectivity estimate is less than $q.minpts$, the ϵ-neighborhood of object p is sparse and can be disregarded. Otherwise, function **RangeQuery** is called to compute the exact number of objects in the ϵ-neighborhood of object p. This way, some sparse ϵ-neighborhoods can be eliminated efficiently.

We observe that a finer grid may improve the quality of the selectivity estimation, potentially avoiding additional expensive **RangeQuery** operations. However, the cost of the selectivity estimation also increases when using a finer grid. In the empirical study, we study how the performance is affected by the granularity of the grid.

3.2.3 FastRange

In addition for supporting selectivity estimation, SGPLs are also useful for processing range queries that are issued on the IR-tree. We propose an algorithm **FastRange** that handles range queries on SGPL. Before presenting the algorithm, we first override the parameterised merging operator $\bigoplus(q_s)$ as $\tilde{\bigoplus}(q_s)$.

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

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 $\tilde{\bigoplus}(q_s)$ maintains the *list of the identifiers* of the objects inside each cell intersecting query range q_s. Function **FastRange** takes two arguments: *list*, the result of operator $\tilde{\bigoplus}(q_s)$, and q_c, a circle centered at an object p 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. If a cell c intersects q_c, only objects in c that have distance to p at most ϵ are added to the result.

4 OPTICS-BASED APPROACH

The DBSCAN-based approach is suitable when users have clear density requirements, i.e., when clear choices exist for parameters ϵ and $minpts$. However, in some cases, it may be difficult to provide such parameters. For example, appropriate values of the parameters may depend on the characteristics of the data. We proceed to present an approach to find the top-k spatial textual clusters without requiring users to provide ϵ and $minpts$ parameters.

We adopt the idea of OPTICS [25] that generates an ordering of the data representing its density-based clustering structure. The clusters are then extracted based on that ordering. Although OPTICS needs the parameters ϵ and $minpts$ when creating an ordering, the result is insensitive to the values of the parameters. In other words, we can hide ϵ and $minpts$ from users and use default values for the two parameters.

Section 4.1 introduces relevant concepts in OPTICS. We extend OPTICS to process k-STC queries in Section 4.2.1. To improve the performance, an index-based algorithm OPTICS-OM is proposed in Section 4.2.2.

4.1 Preliminaries [25]

Definition 9: Let *minpts-dist*(p) be the distance from relevant object p to its $minpts^{th}$ neighbor. The core-distance *c-dist*(p) of p is defined as follows.

$$\textit{c-dist}(p) = \begin{cases} +\infty, & \text{if } |N_\epsilon(p)| < \textit{minpts} \\ \textit{minpts-dist}(p), & \text{otherwise} \end{cases} \quad (5)$$

Definition 10: Let p and o be relevant objects from D_ψ and let $N_\epsilon(o)$ be the ϵ-neighborhood of o. The reachability-distance *r-dist*(p, o) of p with respect to o is defined as follows.

$$\textit{r-dist}(p,o) = \begin{cases} +\infty, \text{if } |N_\epsilon(o)| < \textit{minpts} \\ \max(\textit{c-dist}(o), \|o\ p\|), \text{otherwise} \end{cases} \quad (6)$$

Ordering Generation (OG). The OPTICS algorithm [25] generates an ordering of the objects L and the corresponding reachability values, denoted by $r(p)$, in the following way. A priority queue Q is used to organize the objects to be processed, prioritized by $r(p)$. Initially, a randomly selected object is added to Q, and its reachability value is set to $+\infty$.

1) The next unprocessed object p is appended to L together with $r(p)$; then $N_\epsilon(p)$ and *c-dist*(p) are computed.
2) If p is a core, for each unprocessed object p_i in $N_\epsilon(p)$, we compute *r-dist*(p_i, p). If p_i is not in the priority queue Q, we assign the value of *r-dist*(p_i, p) to $r(p_i)$ and add p_i to Q. Otherwise, we update $r(p_i)$ in Q to be $\min(r(p_i), \textit{r-dist}(p_i, p))$.
3) If p is not a core, we discard it.

Cluster Detection. After computing the ordered list of objects L, the clusters are discovered according to Defintion 13 by scanning the objects in L sequentially. To define the clusters, we need a couple of auxiliary concepts.

Definition 11: An object $p \in D_\psi$ is an ξ-steep upward object if it is $\xi\%$ lower than its successor p^s:

$$\textit{UpPoint}_\xi(p) \iff r(p) \leq r(p^s) \times (1-\xi)$$

An object $p \in D_\psi$ is an ξ-steep downward object if p's successor p^s is $\xi\%$ lower:

$$\textit{DownPoint}_\xi(p) \iff r(p) \times (1-\xi) \geq r(p^s)$$

Definition 12: An interval $I = [s, e]$ is an ξ-steep upward area $\textit{UpArea}_\xi(I)$ if it satisfies the following conditions:

- Both s and e are ξ-steep upward objects, i.e., $\textit{UpPoint}_\xi(s)$ and $\textit{UpPoint}_\xi(e)$.
- Each object x between s and e is at least as high as its predecessor x^p:

$$\forall x, s < x < e\ (r(x) \geq r(x^p))$$

- I contains at most *minpts* consecutive points that are not ξ-steep upward objects.
- I is maximal.

An ξ-steep downward area $\textit{DownArea}_\xi(I)$ is defined analogously.

Definition 13: An interval $C = [s, e]$ is an ξ-cluster, denoted $\textit{cluster}_\xi(C)$ if $\exists D = [s_D, e_D]$ and $U = [s_U, e_U]$ exist that satisfy the following conditions:

1) $\textit{DownArea}_\xi(D) \wedge s \in D$
2) $\textit{UpArea}_\xi(U) \wedge e \in U$
3) $e - s \geq \textit{minpts}$
4) $\forall x \in (e_D, s_U)(r(x) \leq \min\{r(s_D), r(e_U)\} \times (1-\xi))$
5) $(s, e) = \begin{cases} (\max\{x \in D | r(x) > r(e_U^s)\}, e_U), \\ \qquad \text{if } r(s_D) \times (1-\xi) \geq r(e_U^s) \\ (s_D, \max\{x \in U | r(x) < r(s_D)\}), \\ \qquad \text{if } r(e_U^s) \times (1-\xi) \geq r(s_D) \\ (s_D, e_U), \text{otherwise} \end{cases}$

We use the example shown in Figures 7(e) and 7(f) to explain the intuition underlying OPTICS. Suppose that Figure 7(f) shows the output of the ordering generated for the objects shown in Figure 7(e). In this ordered list, the $r(p_i)$ value of an object p_i is its reachability-distance w.r.t. one object located before p_i in the list. Consecutive objects with small $r(p_i)$ values are close to each other. Objects with large $r(p_i)$ values are far from the objects located before them in the list, which can be signs of separation between clusters. The two valleys shown in Figure 7(f) are considered as clusters by OPTICS.

4.2 Algorithms

4.2.1 Baseline OPTICS-OG

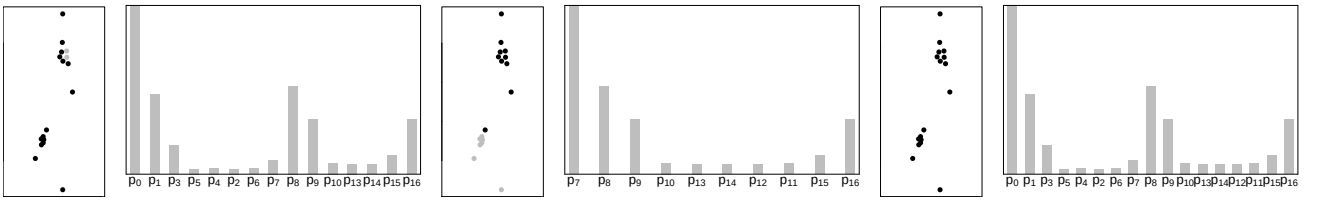
The OPTICS-based baseline algorithm encompasses two phases. In the first, the algorithm uses the inverted index to retrieve all objects D_ψ that are relevant to the query keywords. In the second, it applies algorithm OG to generate an ordering of the objects in D_ψ. Finally, it discovers the clusters according to Definition 13. The clusters are ranked, and the top-k results are returned.

4.2.2 OM Algorithm

The baseline algorithm is inefficient because it calls algorithm OG to compute the ordering of the relevant objects from scratch for each query. We propose an index-based algorithm, called OM (Ordering Merging), that generates and stores the ordering of the objects for each word in advance. When a k-STC query arrives, the orderings of objects of each word in the query are loaded and merged so that the ordering of the relevant objects and the corresponding $r(p)$ used for discovering clusters can be obtained efficiently.

Word-Ordering Index. Let $D(w)$ be the set of objects whose documents contain word w. Algorithm OG is applied to $D(w)$ to obtain an ordered list of objects $L(w)$, where each entry is of the form $(p_i, r(p_i))$. Figures 7(b) and 7(d) show the lists for words w_1 and w_2, respectively. The two valleys in Figure 7(b) indicate two potential clusters of objects that are relevant to w_1. Similarly, the valley in Figure 7(d) indicates one potential cluster of objects that are relevant to w_2.

For each word w, algorithm OG consumes the objects in $D(w)$ in ascending order of the object identifier. This way, objects in different ordered lists that belong to the same spatial region will have consistent identifiers. For instances in Figures 7(a) and 7(c), the objects in the upper spatial region have consistent identifiers in $L(w_1)$ (Figure 7(b)) and $L(w_2)$



(a) $D(w_1)$ (b) $L(w_1)$ (c) $D(w_2)$ (d) $L(w_2)$ (e) D_ψ (f) L_ψ

Fig. 7. Example of Ordering Merging

(Figures 7(d)). The OM algorithm introduced below relied on the consistency of object identifiers.

The word-ordering index consists of the ordered lists $L(w)$ of all the words in the data set. It can be implemented as a key-value mapping, so that $L(w)$ can be retrieved using word w.

Ordering Merging (OM). Having the ordered lists of all the words indexed, OM (Algorithm 3) computes the ordering $L(\psi)$ of the objects relevant to the given query keywords ψ in the following way.

1) For each keyword w_i in ψ, it loads its ordered list $L(w_i)$.
2) If ψ contains only one keyword, $L(w)$ is returned.
3) Otherwise, it merges all ordered lists $L(w_i)$ into one ordered list $L(\psi)$ as follows:
   a) A priority queue $queue$ with key $r(p)$ is used to organize the entries in the ordered lists. Ties are broken using object identifiers.
   b) The first entries are removed from all ordered lists and added to $queue$.
   c) If $queue$ is not empty, the first entry e is dequeued. If the object in this entry has not been seen before, entry e is added to $L(\psi)$. Otherwise, this entry is discarded.
   d) Let $\mathbf{L}(e)$ be the set of ordered lists, where the object in e is removed from each list in the previous step. Next, the current first entry e' that has not been seen before from each ordered list in $\mathbf{L}(e)$ is removed and added to $queue$.
   e) Steps c) and d) are repeated until $queue$ is empty. Finally, $L(\psi)$ is returned.

Given the ordering $L(\psi)$ of the objects relevant to the query keywords, clusters are extracted based on Definition 13. In Figure 7, in the plots ((b), (d), (f)) of an ordering of objects, valleys indicate potential clusters. The ordering produced by algorithm OM differs from that generated by algorithm OG. The clusters extracted from the ordering produced by OM may overlap spatially, due to how lists are merged. Hence, a post-processing step that merges spatially overlapping clusters is required. As will be shown in the case study and efficiency evaluation in Section 5, OG finds clusters that cannot be found by OM, and OG also fails to find clusters discovered by OM. Yet, the quality of the OG and OM clusters (based on DBCV) are similar. But OM incurs substantially lower computational cost than OG.

Example 4.1: Figures 7(a) and 7(c) show the objects whose documents contain w_1 and w_2, respectively. Given $minpts = 2$, $L(w_1)$ contains the corresponding entries of the 15 objects $\{p_0, p_1, p_3, \cdots, p_{16}\}$, and $L(w_2)$ contains the corresponding entries of the 10 objects $\{p_7, p_8, p_9, \cdots, p_{16}\}$. Figures 7(b) and 7(d) show the $r(p_i)$ values of the objects in $L(w_1)$ and $L(w_2)$, respectively. Given $\psi = \{w_1, w_2\}$, after merging $L(w_1)$ and $L(w_2)$, we obtain the list of objects whose documents are relevant to ψ, as shown in Figure 7(f). Specifically, objects p_0, p_1, p_3, $\cdots$, p_6 and their $r(p)$ are added to $L(\psi)$ and removed from $L(w_1)$. When encountering object p_7 in both $L(w_1)$ and $L(w_2)$, the smaller $r(p)$ from $L(w_1)$ is used in $L(\psi)$. □

Algorithm 3 OM(Query q, Index $windex$)

```
1: for each w_i ∈ q.ψ do
2:     L(w_i) ← LoadOrderedList(w_i, windex);
3: if |q.ψ| = 1 then
4:     Return L(w);
5: PriorityQueue queue ← ∅;
6: L(ψ) ← ∅;
7: for each w_i ∈ q.ψ do
8:     Entry e ← L(w_i).RemoveFirst();
9:     queue.Enqueue(e);
10: while queue is not empty do
11:     Entry e ← queue.Dequeue();
12:     if e has not been seen before then
13:         Add e to L(ψ);
14:     for each L ∈ L(e) do
15:         Entry e' ← L.Remove();
16:         queue.Enqueue(e');
17: Return L(ψ);
```

5 Empirical Study

We conduct an empirical study to evaluate our proposals. Section 5.1 presents the datasets, queries, parameters, and platform used in the study. The quality of the clusters found by DBSCAN, OPTICS-OG, and OPTICS-OM is analyzed in Section 5.2. The efficiency of the DBSCAN-based and the OPTICS-based approaches are evaluated in Section 5.3.

5.1 Experimental Setup

We use a subset of SimpleGeo[2] for efficiency evaluation, which roughly covers the entire United States of America. It contains 10,823,427 objects. Each object has a text description that is six words long on average. The total number of distinct words is 33,314. To evaluate the quality of clusters, we extract a subset of SimpleGeo, named as Arizona that roughly covers a spatial rectangular range $[(-114.0, 31.0), (-108.0, 37.0)]$ and contains 216,070 objects.

We generate 4 query sets in the space of the dataset, in which the number of keywords is 1, 2, 3, or 4. Each set comprises 50 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. We then take the location of the object as the query location and randomly choose words from the document of the object as the query keywords.

Table 2 shows the parameter values used in the experiments, where the values in bold are default values. All algorithms were implemented in Java, and an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 100GB main memory was used for the experiments. All the data structures are memory resident. We report the average elapsed time consumed and the average number of range queries issued by the k-STC queries.

TABLE 2
Parameter Values

Parameter	Values
k	5, **10**, 15, 20
$\|q.\psi\|$	1, **2**, 3, 4
ϵ	0.0001, **0.0005**, 0.001, 0.005, 0.01
$minpts$	10, 20, **50**, 100, 200
h	6, 8, 10, **12**, 14, 16
α	0.1, 0.3, **0.5**, 0.7, 0.9

5.2 Qualitative Analysis

Visual Analysis. We compare visually the clusters found by DBSCAN, OPTICS-OG, and OPTICS-OM. The density parameters are determined based on the k-dist plot [32] of the data (shown in Figure 8), in which a sharp change corresponds to a suitable value of ϵ. Figure 9 shows the results found by the three approaches on small regions from datasets Arizona. Generally speaking, DBSCAN finds clusters that cover large spatial regions, while OPTICS-OG and OPTICS-OM can discover small clusters inside the large clusters found by DBSCAN. These small clusters cannot be found by DBSCAN via tuning the density parameters. The objects in these small clusters are either considered as noise when increasing parameter $minpts$ or included in a large cluster when decreasing parameter $minpts$. The clusters found by OPTICS-OM and OPTICS-OG are similar. The following case study illustrates the differences of these approaches.

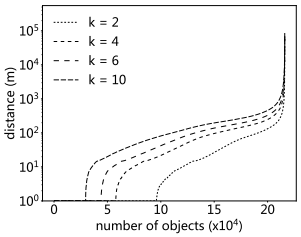


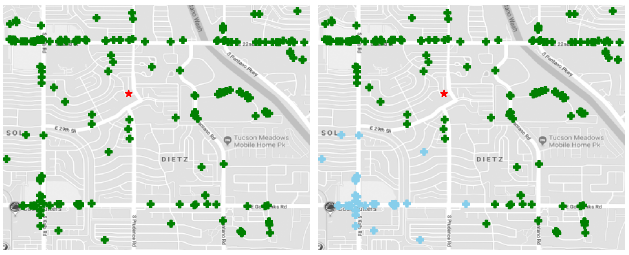
Fig. 8. k-dist Plot (Arizona)

In Figure 9, the query keywords are "retail goods," and the query location is given by symbol "★". DBSCAN finds a large cluster surrounding the query location—see Figures 9(a) and 9(b). The query user cannot get useful information from these results and does not know where to go, especially when the user does not have time to explore a large region. OPTICS-OG and OPTICS-OM return small clusters close to the query location, e.g., the cluster represented by green dots at the top center in Figure 9(c) and the cluster represented by red triangles in Figure 9(d). With these results, the query user is able to choose a small region, where relevant objects can be found.

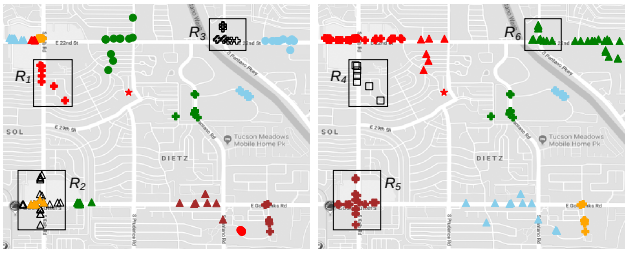
To understand the differences between OPTICS-OG and OPTICS-OM in this case study, we show the clusters found by OPTICS-OG for keywords "goods" and "retail" in Figures 9(e) and 9(f), respectively. Comparing the results of OPTICS-OG and OPTICS-OM, we have the following two observations. *First*, OPTICS-OM cannot recognize the objects in region R_4 as a cluster (Figure 9(d)), while OPTICS-OG finds a cluster in the same region, denoted by R_1 (Figure 9(c)). The reason is that the set of objects whose documents contain "goods" in that region, denoted by R_7 in Figure 9(e), does not satisfy the conditions of forming a cluster. Neither does the set of objects whose documents contain "retail," denoted by R_{10} in Figure 9(f). But the union of the objects in R_7 and R_{10} yields a dense region. Thus, based on the ordered lists of "goods" and "retail," OPTICS-OM considers them as noise, while OPTICS-OG finds a cluster in R_1 based on the union of the objects. *Second*, OPTICS-OG fails to find the cluster in region R_3 (Figure 9(c)) while OPTICS-OM does, denoted by R_6 (Figure 9(d)). The reason is as follows. The set of objects whose documents contain "goods" in region R_9 (Figure 9(e)) is considered as a cluster, while the set of objects whose documents contain "retail" in the same region, denoted by R_{12} (Figure 9(f)), is considered as noise. Because of the merging method in OPTICS-OM, the cluster of objects in R_9 is kept, denoted by R_6 in Figure 9(d). However, the union of the objects in R_9 and R_{12} forms a region where separations exist among small sets of objects. OPTICS-OG considers this layout as noise. For the same reason, OPTICS-OG

2. https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces

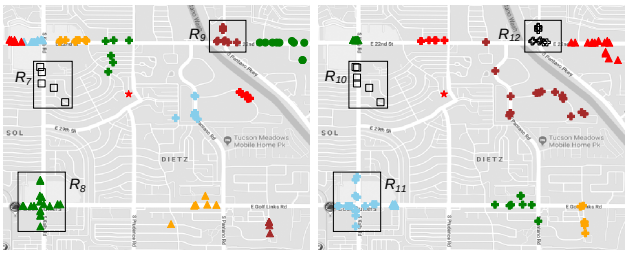
and OPTICS-OM find different clusters at the bottom left, denoted by R_2 and R_5.



(a) DBSCAN ($minpts = 5$) (b) DBSCAN ($minpts = 20$)



(c) OPTICS-OG ($minpts = 5$, $\xi = 0.01$) (d) OPTICS-OM ($minpts = 5$, $\xi = 0.01$)



(e) OPTICS-OG ("goods," $minpts = 5$, $\xi = 0.01$) (f) OPTICS-OG ("retail," $minpts = 5$, $\xi = 0.01$)

Fig. 9. Arizona ($\epsilon = 552$ m)

Cluster Quality Evaluation. We evaluate the quality of the clusters found by DBSCAN, OPTICS-OG, and OPTICS-OM using DBCV [33], a relative validation index for density-based, arbitrarily shaped clusters. The index assesses clustering quality based on the relative density connections between pairs of objects. The larger the index value, the better the clusters. We extract a set of objects from SimpleGeo by arbitrarily specifying a spatial rectangular range $[(-113.85, 38.61), (-107.93, 41.85)]$, which contains 99,272 objects. Table 3 reports the DBCV index of the three approaches on the extracted set of objects when varying the number of query keywords from 1 to 4. In each test, 10 queries are processed, and the average DBCV index is calculated. The results show that OPTICS-OG and OPTICS-OM outperform DBSCAN with various values of $minpts$ and that OPTICS-OM and OPTICS-OG produces similar results. In addition, OPTICS-OM finds fewer clusters than does OPTICS-OG, which is expected according to the ordering merging algorithm in OPTICS-OM. The bottom part of Table 3 shows the number of clusters and the average cluster sizes discovered by OPTICS-OG and OPTICS-OM.

TABLE 3
DBCV Index ($\epsilon = 1003$ m)

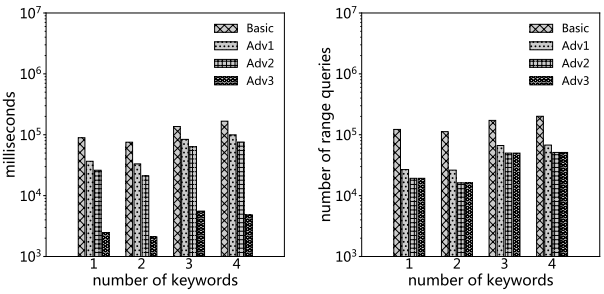
# of query keywords	1	2	3	4	Average
DBSCAN ($minpts = 5$)	-0.066	-0.379	-0.416	-0.652	-0.378
DBSCAN ($minpts = 10$)	-0.059	-0.292	-0.318	-0.604	-0.318
DBSCAN ($minpts = 15$)	-0.020	-0.228	-0.260	-0.541	-0.262
OPTICS-OG ($minpts = 5$, $\xi = 0.01$)	0.126	0.109	0.113	0.125	0.118
OPTICS-OM ($minpts = 5$, $\xi = 0.01$)	0.126	0.108	0.115	0.097	0.112
Number of clusters					
OPTICS-OG ($minpts = 5$, $\xi = 0.01$)	290	756	663	1604	-
OPTICS-OM ($minpts = 5$, $\xi = 0.01$)	290	673	611	1337	-
Average size of clusters					
OPTICS-OG ($minpts = 5$, $\xi = 0.01$)	9	10	10	10	-
OPTICS-OM ($minpts = 5$, $\xi = 0.01$)	9	10	11	12	-

5.3 Performance Evaluation

5.3.1 Algorithms of DBSCAN-based Approach

We evaluate the performance of the basic approach (Basic) and the advanced approaches with selectivity estimation (Adv1), with selectivity estimation and object skipping (Adv2), and with selectivity estimation, object skipping, and FastRange (Adv3) under varying parameter settings. Overall, after applying the three enhancements, the performance of the basic approach is improved 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 reduces the cost of the range queries.

Varying the Number of Keywords $|q.\psi|$**.** Figure 10 shows the performance of the four approaches when varying the number of query keywords. Their performance get worse as the number of query keywords increases, since more objects are involved and the search space becomes larger. As expected, the number of range queries issued (Figure 10(b)) is consistent with the runtime (Figure 10(a)). In the rest of the experiments, we only report the runtime.



(a) Runtime (b) # of range queries

Fig. 10. Varying Number of Keywords

Varying Density Parameters. The density requirement involves two parameters, ϵ and $minpts$. Figure 11 shows the performance of the four approaches when varying ϵ. Figure 12 shows their performance when varying $minpts$. Large ϵ and small $minpts$ indicate a low density requirement. In general, as ϵ ($minpts$) increases (decreases), the performance becomes worse. The reason is that low density requirement makes 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 increase the computational cost. The performance gap between Basic and Adv1 (using selectivity estimation) is large when the density requirement is high (using small ϵ or large $minpts$). This is because many objects become noises and are pruned by the selectivity estimation, so that the number of range queries is reduced. On the other hand, the performance gap between Adv1 and Adv2 is large when the density requirement is low. Here, the reason is that many objects become cores and cannot be pruned by selectivity estimation, but are pruned by object skipping. Hence, selectivity estimation is effective when the density requirement is high, while object skipping is effective when the density requirement is low.

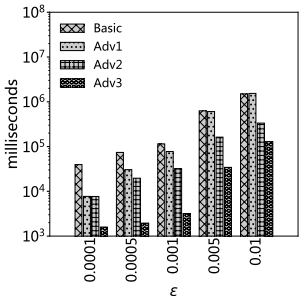


Fig. 11. Varying ϵ

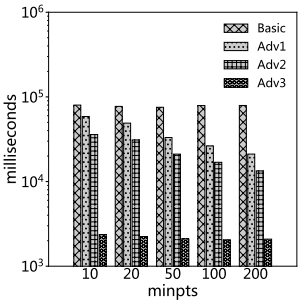


Fig. 12. Varying $minpts$

Varying α. Figure 13 shows the runtimes of the four approaches when varying α that balances the spatial distance and the text relevance in the ranking function. Their performance is insensitive to parameter α. Hence, our advanced approach offers good performance no matter what the user preferences are.

Varying the Number of Requested Clusters k. Figure 14 shows the runtimes when varying the number of requested clusters k. 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 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 15 shows how the performance of the basic and advanced algorithm change as the size of the data set increases. The data sets used in this experiment are randomly generated from SimpleGeo. We observe that both algorithms scale nearly linearly. The advanced algorithm significantly outperforms the basic algorithm.

Varying the Order of Z-Curve h in SGPL. The SGPL is constructed based on a space filling curve indexed grid over the dataset. We adopt the Z-curve in our evaluation. Other space filling curves are also applicable. The order h of the Z-curve defines the granularity of the grid, which in turn affects the performance of the SGPL. Large h provide finer granularities, so that the selectivity estimation of the range queries for detecting sparse ϵ-neighborhoods is improved. The results in Figure 16 illustrate this point. As h increases, the performance improves. We observe that when h exceeds 12, the performance of the SGPL gets worse. This is because it takes time to process more grid cells. This indicates that it is not beneficial to construct an SGPL using a finer grid.

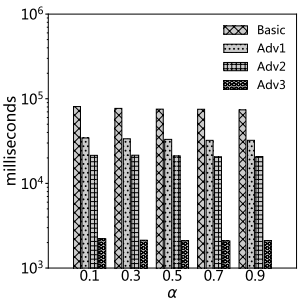


Fig. 13. Varying α

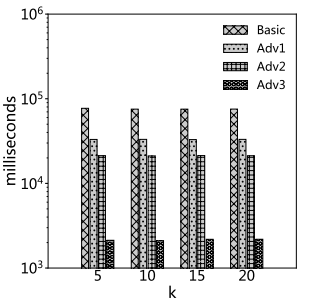


Fig. 14. Varying k

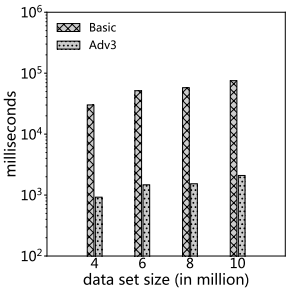


Fig. 15. Varying Data Size

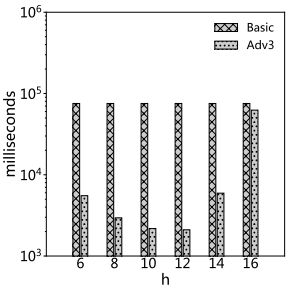


Fig. 16. Varying h

5.3.2 Algorithms of OPTICS-based Approach

We proceed to evaluate the performance of algorithms OPTICS-OG and OPTICS-OM in different parameter settings. According to the k-dist plot of SimpleGeo (Figure 17), the density parameters are set as follows: $\epsilon = 1152m$, $minpts = 4$, $\xi = 0.01$.

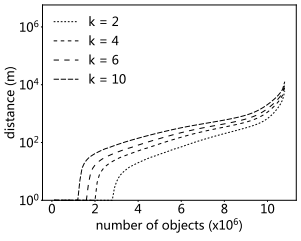


Fig. 17. k-dist Plots on SimpleGeo

Varying the Number of Keywords $|q.\psi|$. Figure 18 shows the performance of the two approaches when

varying the number of query keywords. Their performance decreases as the number of query keywords increases. This is because the number of objects involved in computation increases. OPTICS-OM significantly outperforms OPTICS-OG, and the performance gap grows with the number of query keywords. The reason is that when the number of keywords is large, there are many relevant objects, and OPTICS-OM is designed for saving computations on large sets of objects.

Varying α. Figure 19 shows the runtime of the two approaches when varying α that balances the spatial distance and the text relevance in the ranking function. Their performance is insensitive to parameter α. OPTICS-OM beats OPTICS-OG for all values of α. Hence, OPTICS-OM algorithm can guarantee a good performance no matter what the user preferences are.

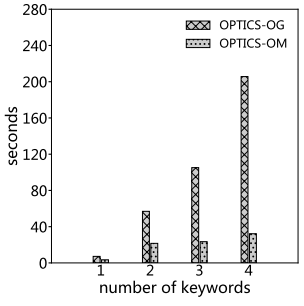


Fig. 18. Varying $|q.\psi|$

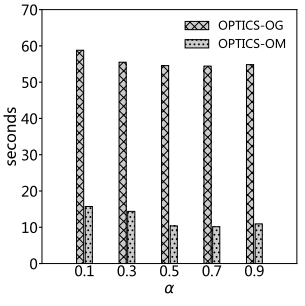


Fig. 19. Varying α

Varying the Number of Requested Clusters k. Figure 20 shows the runtime of the two approaches when varying the number of requested clusters k. Their performance is insensitive to parameter k. Again, OPTICS-OM performs significantly better than OPTICS-OG for all values of k.

Scalability. Figure 21 shows how the performance of OPTICS-OG and OPTICS-OM change as the size of the data set increases. The data sets used are randomly generated from SimpleGeo. We observe that both algorithms scale linearly.

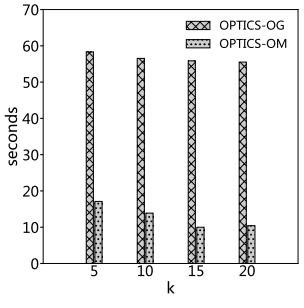


Fig. 20. Varying k

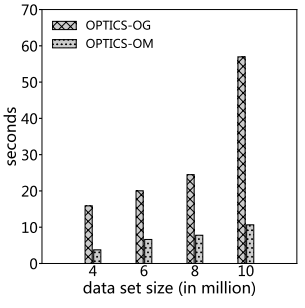


Fig. 21. Varying Data Size

6 Related Work

Spatial Textual Search. Spatial keyword search has attracted much attention in recent years. A spatial keyword query retrieves spatial web objects (e.g., web content relating to shops and restaurants) that are spatially and textually relevant to provided query arguments. Several efficient geo-textual indexes have been proposed to support spatial keyword queries. Most indexes combine the R-tree for spatial indexing with inverted or signature files for text indexing, such as the IR^2-tree [3], the IR-tree [10], and the S2I [8]. These hybrid structures are able to utilize both spatial and textual information to prune the search space at query time. Our proposal is orthogonal to these indexes. We adopt the state-of-art IR-tree [10] in the basic algorithm. Other R-tree based indexes can also be used.

Beyond indexing, many studies investigate interesting variants of the typical spatial keyword query that satisfy different user needs. Chen et al. [34] study the problem of matching a stream of incoming Boolean range continuous queries over a stream of incoming geo-textual objects in real time. The keyword-aware optimal route query [35] finds an optimal route that covers a set of user-specified keywords and satisfies a specified budget constraint such that an objective score of the route is optimal. Collective spatial keyword search [15], [36], [37] aims to retrieve a group of objects that are close to a query point and collectively cover a set of query keywords. Wu et al. [9] explore the problem of maintaining the result set of top-k spatial keyword queries while the user is moving. Salgado et al. [38] propose an efficient solution for processing continuous range spatial keyword queries over moving spatio-textual objects. In the spatial keyword search, some studies [14], [39] take the objects nearby into account when ranking the results. The Reverse Spatial Textual k Nearest Neighbor (RSTkNN) query [12] finds objects that take the query object as one of their k most spatial-textual similar objects. Bøgh et al. [40] 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 specific product. A spatio-textual similarity join [41] retrieves pairs of objects that are spatially close and textually similar. Zhang et al. [42], [43] study a query that takes a set of keywords as parameters and returns 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. A semantic-aware top-k spatial keyword query [44] returns the k objects most similar to the query, subject to not only their spatial and textual properties, but also the coherence of their semantic meanings. Zhang et al. [45] augment the spatial keyword search with a boolean expression constraint, which is used to query structured or semi-structured spatial entities.

These studies either return single objects as results

or retrieve a set of objects located in an ad-hoc spatial region (e.g., a rectangular region). Our work aims to discover natural clusters of objects that are relevant to a given query.

The paper substantially extends a 6-page conference paper [46] that proposed the DBSCAN-based approach to computing the k-STC query.

Density-based Clustering Algorithms. In density-based clustering, clusters are defined as regions of higher density than the remainder of the data set. Objects in the sparse regions that are required to separate clusters are often considered to be noise and border points. The most popular density-based clustering method is DBSCAN [24]. It features a well-defined cluster model called"density-reachability." It is based on connecting points that are within a certain distance range of each other. A cluster consists of all density-connected objects, plus all objects that are within these objects' range. OPTICS [25] is a generalization of DBSCAN that removes the need to choose an appropriate value for the range parameter, and produces a hierarchical result related to that of linkage clustering. DBRS [47] improves DBSCAN by repeatedly picking an unclassified point at random and examining its neighborhood. VDBSCAN [32] supports varied-density dataset clustering. Before adopting traditional DBSCAN algorithm, some method is 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 varying densities simultaneously. GDBSCAN [48] clusters spatial objects according to both their spatial and nonspatial attributes.

These clustering models can be considered as components in our work. We use two state-of-the-art methods, i.e., DBSCAN and OPTICS, in the k-STC query.

7 Conclusions

This paper proposes a new type of query, the top-k spatial textual cluster retrieval (k-STC) query that returns the top-k clusters, such that (i) they are located close to a query location, (ii) they contain objects whose text descriptions are relevant to query keywords, and (iii) the density of the clusters satisfies a query constraint. We propose DBSCAN-based and OPTICS-based approaches for evaluation of the k-STC query. The DBSCAN-based approach is suitable when users have clear density preferences. It 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 range queries so that sparse neighborhoods can be detected quickly, and (iii) a fast range query algorithm on the SGPL. The OPTICS-based approach is preferable when it is difficult to set the density parameters of DBSCAN. It includes a word-ordering index and efficient ordering merging algorithm OPTICS-OM. An empirical study on real datasets provide evidence that the paper's proposals are effective at finding good quality clusters and at offering scalable performance.

Acknowledgement

This work was supported in part by grants No. 2019A1515011721 and No. 2019A1515011064 from Natural Science Foundation of Guangdong Province, China.

References

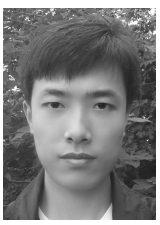
[1] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *SIGMOD*, 2006, pp. 277–288.
[2] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.
[3] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 656–665.
[4] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *SSDBM*, 2007, p. 16.
[5] A. Khodaei, C. Shahabi, and C. Li, "Hybrid indexing and seamless ranking of spatial and textual features of web documents," in *DEXA*, 2010, pp. 450–466.
[6] 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.*, vol. 23, no. 4, pp. 585–599, 2011.
[7] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *CIKM*, 2005, pp. 155–162.
[8] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *SSTD*, 2011, pp. 205–222.
[9] D. Wu, M. L. Yiu, and C. S. Jensen, "Moving spatial keyword queries: Formulation, methods, and analysis," *ACM Trans. Database Syst.*, vol. 38, no. 1, p. 7, 2013.
[10] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *VLDB J.*, vol. 21, no. 6, pp. 797–822, 2012.
[11] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1889–1903, 2012.
[12] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *SIGMOD*, 2011, pp. 349–360.
[13] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *ICDE*, 2011, pp. 541–552.
[14] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *PVLDB*, vol. 3, no. 1, pp. 373–384, 2010.
[15] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD*, 2011, pp. 373–384.
[16] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *EDBT*, 2012.
[17] T. Abeywickrama, M. A. Cheema, and A. Khan, "K-SPIN: efficiently processing spatial keyword queries on road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 5, pp. 983–997, 2020.
[18] G. Li, J. Feng, and J. Xu, "Desks: direction-aware spatial keyword search," in *ICDE*, 2012.
[19] S. N. Durlauf and L. E. Blume, *The New Palgrave Dictionary of Economics*, 2nd ed. Palgrave Macmillan, 2008.
[20] D.-W. Choi, C.-W. Chung, and Y. Tao, "A scalable algorithm for maximizing range sum in spatial databases," *PVLDB*, vol. 5, no. 11, pp. 1088–1099, 2012.
[21] Y. Tao, X. Hu, D.-W. Choi, and C.-W. Chung, "Approximate maxrs in spatial databases," *PVLDB*, vol. 6, no. 13, pp. 1546–1557, 2013.

[22] J. Liu, G. Yu, and H. Sun, "Subject-oriented top-k hot region queries in spatial dataset," in *CIKM*, 2011, pp. 2409–2412.
[23] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu, "Retrieving regions of interest for user exploration," *PVLDB*, vol. 7, no. 9, pp. 733–744, 2014.
[24] 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*, 1996, pp. 226–231.
[25] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," in *SIGMOD*, 1999, pp. 49–60.
[26] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, pp. 613–620, 1975.
[27] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR*, 1998, pp. 275–281.
[28] J. Zobel and A. Moffat, "Inverted files for text search engines," in *ACM Comput. Surv.*, vol. 38, no. 2, 2006, p. 6.
[29] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.
[30] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proc. VLDB Endow.*, vol. 6, no. 3, pp. 217–228, 2013.
[31] J. K. Lawder and P. J. H. King, "Using space-filling curves for multi-dimensional indexing," in *BNCOD*, 2000, pp. 20–35.
[32] P. Liu, D. Zhou, and N. Wu, "VDBSCAN: Varied density based spatial clustering of applications with noise," in *Service Systems and Service Management*, 2007, pp. 1–4.
[33] D. Moulavi, P. A. Jaskowiak, R. J. G. B. Campello, A. Zimek, and J. Sander, "Density-based clustering validation." in *SDM*. SIAM, 2014, pp. 839–847.
[34] L. Chen, G. Cong, and X. Cao, "An efficient query indexing mechanism for filtering geo-textual data," in *SIGMOD*, 2013, pp. 749–760.
[35] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *PVLDB*, vol. 5, no. 11, pp. 1136–1147, 2012.
[36] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: a distance owner-driven approach," in *SIGMOD*, 2013, pp. 689–700.
[37] H. K. Chan, C. Long, and R. C. Wong, "On generalizing collective spatial keyword queries," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1712–1726, 2018.
[38] C. Salgado, M. A. Cheema, and M. E. Ali, "Continuous monitoring of range spatial keyword query over moving objects," *World Wide Web*, vol. 21, no. 3, pp. 687–712, 2018.
[39] V. J. Wei, R. C. Wong, C. Long, and P. Hui, "On nearby-fit spatial keyword queries (extended abstract)," in *ICDE*. IEEE, 2020, pp. 2024–2025.
[40] A. Skovsgaard and C. S. Jensen, "Finding top-k relevant groups of spatial web objects," *VLDB J.*, vol. 24, no. 4, pp. 537–555, 2015.
[41] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-textual similarity joins," *PVLDB*, vol. 6, no. 1, pp. 1–12, 2012.
[42] 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*, 2009, pp. 688–699.
[43] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *ICDE*, 2010, pp. 521–532.
[44] Z. Qian, J. Xu, K. Zheng, P. Zhao, and X. Zhou, "Semantic-aware top-k spatial keyword queries," *World Wide Web*, vol. 21, no. 3, pp. 573–594, 2018.
[45] D. Zhang, Y. Li, X. Cao, J. Shao, and H. T. Shen, "Augmented keyword search on spatial entity databases," *VLDB J.*, vol. 27, no. 2, pp. 225–244, 2018.
[46] D. Wu and C. S. Jensen, "A density-based approach to the retrieval of top-k spatial textual clusters," in *CIKM*. ACM, 2016, pp. 2095–2100.
[47] X. Wang and H. J. Hamilton, "DBRS: A density-based spatial clustering method with random sampling," in *PAKDD*, 2003, pp. 563–575.
[48] 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.*, vol. 2, no. 2, pp. 169–194, 1998.

Dingming Wu is Associate Professor of College of Computer Science & Software Engineering at Shenzhen University, China. She received her Bachelor degree in Computer Science at Huazhong University of Science and Technology, Wuhan, China in 2005, and a Master degree in Computer Science at Peking University, Beijing, China in 2008. She received a Ph.D. degree in Computer Science at Aalborg University, Denmark in 2011. Her research concerns data management, query processing, information retrieval, and data mining.

Ilkcan Keles is an expert software developer at Turkcell. He was an assistant professor at Aalborg University before joining Turkcell. He obtained his PhD degree from the Dept. of Computer Science, Aalborg University in 2018. He worked on the evaluation of ranking functions used for spatial keyword queries during his PhD studies. He got his MS and BS degrees from Middle East Technical University in 2014 and 2011, respectively.

Song Wu received the bachelor's degree from the School of Software, TianJin University, in 2018. He is now a postgraduate student with the Big Data Lab in ShenZhen University. His research interests include geo-spatial data mining and query processing on spatial-textual data.



Hao Zhou is currently a Master student with the College of Computer Science and Software Engineering, ShenZhen University, China. His research interests include query processing on spatial-textual data and geosocial network mining.

Simonas Saltenis is Associate Professor at the Department of Computer Science, Aalborg University, Denmark as well as Research Professor at the Institute of Computer Science, Vilnius University, Lithuania. His research concerns spatial and spatiotemporal data management. Simonas has a Ph.D. in Computer Science from Aalborg University, Denmark.

Christian S. Jensen is Professor of Computer Science at Aalborg University, Denmark. His research concerns primarily temporal and spatio-temporal analytics, including machine learning, data mining, and query processing. Christian is an ACM and IEEE Fellow, and he is a member of Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several awards for his research, most recently the 2019 IEEE TCDE Impact Award.

Kezhong Lu is Professor of College of Computer Science & Software Engineering at Shenzhen University, China. He received his Bachelor degree and Ph.D. degree in Computer Science at University of Science and Technology of China in 2001 and 2006. His research concerns big data, parallel and distributed computing, algorithms, wireless sensor network, and computational geometry. He is a vice dean of College of Computer Science & Software Engineering and the leader of computer technology area at Shenzhen University, China.