2014 IEEE 15th International Conference on Mobile Data Management

Efficient Top-k Spatial Locality Search for Co-located Spatial Web Objects

Qiang Qu* Siyuan Liu† Bin Yang* Christian S. Jensen‡
* Department of Computer Science, Aarhus University, Denmark
† Heinz College, Carnegie Mellon University, USA
‡ Department of Computer Science, Aalborg University, Denmark
Email: {qu, byang}@cs.au.dk*, siyuan@cmu.edu†, csj@cs.aau.dk‡

***Abstract*—In step with the web being used widely by mobile users, user location is becoming an essential signal in services, including local intent search. Given a large set of spatial web objects consisting of a geographical location and a textual description (e.g., online business directory entries of restaurants, bars, and shops), how can we find sets of objects that are both spatially and textually relevant to a query? Most of existing studies solve the problem by requiring that all query keywords are covered by the returned objects and then rank the sets by spatial proximity. The needs for identifying sets with more textually relevant objects render these studies inapplicable. We propose localitySearch, a query that returns top-k sets of spatial web objects and integrates spatial distance and textual relevance in one ranking function. We show that computing the query is NP-hard, and we present two efficient exact algorithms and one generic approximate algorithm based on greedy strategies for computing the query. We report on findings from an empirical study with three real-life datasets. The study offers insight into the efficiency and effectiveness of the proposed algorithms.**

I. Introduction

Users usually report their geographical locations by geo-enabled devices (e.g., smartphones and tablets), which is an important signal in many mobile and web services. We are witnessing a development where increasing volumes of web content is being associated with location, yielding what we term spatial web objects that have a location and a text description [1]. Examples include web pages that represent places such as shops, bars, and restaurants; microblogging entries with locations; and location-based social network postings [2]. It has been reported that 25% of tweets from mobile devices are associated with locations [3]. One of the most important services on the web is keyword search. Billions of search queries are processed each day. A substantial fraction of such queries have local intent, meaning that they target web content that relates to places near the user. Studies report that 20% of Google desktop searches and 53% of mobile Bing searches have local intent [3].

This entire development gives prominence to spatial keyword queries. Such a query typically takes a location and user-supplied keywords as arguments and returns objects that are not only near the query location, but are also textually relevant to the query keywords. Motivated by such applications, we formulate a query that enables spatial locality search for co-located spatial web objects, called localitySearch. Informally, localitySearch enables a user to find the top-k sets of co-located spatial web objects that are near a query location and are the most relevant to given query keywords.

Several studies are linked with the problem. In spatial keyword querying, we are aware of spatial keyword queries [3], [4], [5], [6]. Most of these return a single set of objects, and they evaluate sets by spatial proximity. For textual descriptions, the existing studies often use Boolean conditions that require a candidate set to cover all the query keywords [4], [6]. The needs for identifying sets with more relevant objects render the Boolean method inapplicable. This paper provides an approach to top-k querying that also considers the textual relevance in set evaluation. The returned results as the k most relevant sets in both the spatial and textual dimensions. In particular, we provide two spatial metrics: 1) pairwise distance in order to measure the spatial similarity of a set of objects; 2) distance between a set of objects and a query location.

Our contribution is fivefold. i) We formulate localitySearch and prove it NP-hard. ii) In terms of object locations, we provide a basic method that returns exact top-k results. iii) To improve the basic method, we suggest to index objects by their locations and textual descriptions. As an example to explain our techniques, we use a grid cell index for simplicity. With indexing structures on the objects, we derive approximate bounds on grid cells and integrate them into the search to prune the search space. iv) We propose a generic approximate algorithm, which can be applied with different greedy strategies. v) We conduct extensive experiments on three real-life datasets to elicit pertinent design properties of the provided algorithms, including efficiency and effectiveness.

The rest of the paper is organized as follows. The problem is formally stated in Section II. Section III provides exact and approximate solutions to solve the5 problem. An experimental study is reported in Section IV. We review the state of the art in Section V, and we conclude the paper in Section VI.

II. Problem Statement

We formalize the problem setting and the localitySearch query in the section. The query aims to find the top-k sets of spatial web objects co-locating within a diameter τ bounded spatial range.

The localitySearch query takes as argument a finite set of spatial web objects: $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ in $\mathbb{R}^2$. An object $o_i \in \mathcal{O}$ thus has a geographical location μ, and also a textual description ψ, i.e., a set of textual keywords.

Before we formulate localitySearch, we introduce three terms that are used to evaluate how good a candidate set R of

978-1-4799-5705-7/14 $31.00 © 2014 IEEE
DOI 10.1109/MDM.2014.39

objects is. The three terms measure 1) the distance between a set and a query location, 2) the textual similarity between a set of spatial web objects and a set of query keywords, and 3) the co-locality or spatial similarity between any two objects in a set. Specifically, $\Theta(R, Q.\mu)$ captures the spatial proximity between a set R and the query location $Q.\mu$; $\Omega(R, Q)$ denotes the textual relevance of the spatial web objects in R to $Q.\psi$; and $\Im(R)$ measures the mutual average discrepancy of the objects contained by R. Weight parameters α and β take values in the range $[0, 1]$, and $\alpha + \beta < 1$. The weights control the relative importance of the three terms. To keep the same magnitude, each term is normalized into the range $[0, 1]$. In the following, we elaborate on each term.

We prefer sets of objects that are close to the query location, and a *spatial proximity function* $\Theta(R, Q.\mu)$ measures the distance between the set R of objects and the query location $Q.\mu$. Without loss of generality, we let $\Theta(R, Q.\mu)$ return the distance between the query location and the closest object in R: $\min_{o_i \in R}(dist(o_i.\mu, Q.\mu))$ where $dist(\cdot, \cdot)$ computes the Euclidean distance between its arguments. To normalize the distance, we divide the distance by the maximum possible Euclidean distance of the region in $\mathbb{R}^2$ covering $\mathcal{O}$, denoted as $dist_{max}$. Then we have the following function:

$$\Theta(R, Q.\mu) = 1 - \frac{\min_{o_i \in R}(dist(o_i.\mu, Q.\mu))}{dist_{max}}. \quad (1)$$

Objects have textual descriptions, and localitySearch returns a set of objects that are relevant to given query keywords. We thus need a *textual relevance function* $\Omega(R, Q)$ to compute the total relevance of a set of objects to the query keywords. Different from Boolean conditions [4], [6], $\Omega(R, Q)$ is able to identify sets with more textually relevant objects. Although several definitions of this function are possible [7], we use the vector space model (*VSM*) [8], which is a well-established information retrieval model. We define the textual relevance of an object set to the query keywords as the total textual relevance divided by a factor $\wp(\tau)$ for normalization:

$$\Omega(R, Q) = \frac{\sum_{o_i \in R} VSM(o_i.\psi, Q.\psi)}{\wp(\tau)}, \quad (2)$$

where $\wp(\tau)$ returns the maximum number of relevant objects within a spatial range with diameter τ.

We propose a *mutual average discrepancy function* $\Im(R)$ to measure the "co-locality" between the objects in a set. For uniformity, we use τ as a factor to normalize the distance. The function is thus defined based on the average mutual distance as:

$$\Im(R) = 1 - \frac{\sum_{1 \le i < j \le |R|} dist(o_i.\mu, o_j.\mu)}{\tau \cdot \binom{|R|}{2}}. \quad (3)$$

We proceed to present the definition of the localitySearch query that returns k subsets of spatial web objects in $\mathcal{O}$.

*Definition 1 (**localitySearch**):* The localitySearch query $Q = \langle \mu, \psi, \tau \rangle$ takes three parameters, where μ is a query location, ψ is a query keyword set, and τ is a diameter threshold to define the co-locality that is the maximal distance between two objects. The number of objects in a result set R, namely $|R|$, is required to exceed 1. A result set R must satisfy the query keywords, meaning that it has to cover all the query keywords in $Q.\psi$, i.e., $\forall o_i \in R\ (o_i.\psi \cap Q.\psi \neq \emptyset) \wedge \cup_{o_i \in R} o_i.\psi \supseteq Q.\psi$. Among all possible such sets, the k sets with the highest scores according to the following scoring function are returned.

$$\xi(R, Q) = \alpha\Theta(R, Q.\mu) + \beta\Omega(R, Q) + (1 - \alpha - \beta)\Im(R) \quad (4)$$

The summation form of the scoring function has been widely used in previous studies [3], [7], [8], [4]. This function can be extended to support other terms. Our techniques are applicable to other kinds of terms that have similar mathematical properties. For instance, the *spatial proximity function* $\Theta(R, Q.\mu)$ can be replaced by network distance; the *textual relevance function* $\Omega(R, Q)$ can be replaced by frequency or numerical support (e.g., Facebook Likes); and the *mutual average discrepancy function* $\Im(R)$ can be used for social similarity or pairwise diversity.

Despite the simplicity of Equation 4, computing the localitySearch query is expensive. Given a localitySearch query, the objective function can be regarded as a binary non-linear function where the variables represent objects and are set to 0 or 1, capturing their existence in a result set. Since solving a non-linear integer programming problem is often NP-hard, and it is likely that the proposed problem is NP-hard as well. We confirm this in Theorem 1.

Theorem 1: The localitySearch query is NP-hard ($0 < \alpha + \beta < 1$, $\alpha, \beta \neq 0$).

Proof: We reduce the maximal independent set problem in graphs to an instance of the localitySearch problem. Here, we assume the simplest case where any subset of objects/vertices covers the query keywords.

In the decision version of localitySearch, given a set of objects $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ and a value $\tilde{\xi}$, we ask whether or not a subset exists with a score larger than $\tilde{\xi}$, while the distance between any two objects in the subset does not exceed diameter threshold τ.

An instance of the independent set problem is defined as follows: given an undirected graph $G = (V, E)$ and an integer k, determine whether G contains an independent set I ($I \subseteq V$) of size larger than k that satisfies the condition that the vertices are not connected, i.e., $\forall i, j \in I(e(i, j) \notin E)$.

We assign weights to all pairs of vertices in graph G. For connected vertices, each edge of a pair is assigned a weight of at least τ, and unconnected pairs are connected with a virtual edge that has a weight less than τ. The assigned weights in the new graph satisfy the triangle inequality and are regarded as distances between vertices in an instance of localitySearch where each vertex is an object. We set $\tau < 1/\tilde{\xi}$ in the instance. Each vertex in the graph also has a weight denoting the textual relevance to the query keywords $Q.\psi$ that is τ. We set $k < \lfloor \frac{\tilde{\xi}-2}{\tau} \rfloor \times |\mathcal{O}|$, where $\lfloor \frac{\tilde{\xi}-2}{\tau} \rfloor \times |\mathcal{O}|$ represents the smallest value of k when the score is $\tilde{\xi}$ according to Equation 4. The reduction is in polynomial time as we assign $\binom{n}{2}$ edges. A solution exists for the instance of independent sets if there is a solution for localitySearch, since connected vertices with $e(i, j) \in E$ cannot be contained in the solution to localitySearch instance under the constraint $dist(i, j) < \tau$. Additionally, $k' > k$, as k is less than the smallest value $\lfloor \frac{\tilde{\xi}-2}{\tau} \rfloor \times |\mathcal{O}|$. Since the average distance is always smaller

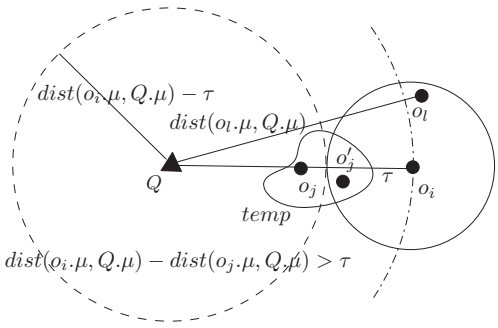


Fig. 1. Generating New Candidate Sets Based on Lemma 1.

than τ (note that $\tau < 1/\tilde{\xi}$), the score ξ of the localitySearch solution is larger than $\tilde{\xi}$ according to Equation 3. Thus, G contains an independent set with size larger than k, which satisfies that $\forall i, j \in I(e(i,j) \notin E)$. The reduction implies that if we had a polynomial time algorithm for localitySearch then we would have a polynomial time algorithm for the independent set problem, which would imply that P=NP. We therefore have that localitySearch is NP-hard. ■

III. Solution to LocalitySearch

We aim to provide efficient algorithms to return exact top-k sets to answer the localitySearch query.

A. Baseline Algorithm

A naive method to process a localitySearch query is to compute the ranking score (according to Equation 4) of each candidate in the power set $2^{\mathcal{O}}$, and return the k sets with the highest ranking scores. The naive method is inefficient as to search the whole space of $2^{\mathcal{O}}$ sets is expensive.

We propose an exact baseline algorithm (BA) such that not every set in $2^{\mathcal{O}}$ is processed. The results of the baseline algorithm are guaranteed to be correct.

Considering an algorithm that examines the objects in $\mathcal{O}$ from near to far according to a query location $Q.\mu$, Lemma 1 is established before we present the algorithm BA.

Lemma 1: Given the i-th nearest neighbor o_i with respect to the query location $Q.\mu$, a candidate object set $temp$ can be removed from the whole candidate set Map if there exists an object $o_j \in temp$ that satisfies $dist(o_i.\mu, Q.\mu) - dist(o_j.\mu, Q.\mu) > \tau$.

Proof: A candidate set $temp$ can be safely removed from Map if and only if $temp$ with an object that is going to be examined in the following steps of the algorithm cannot become a new candidate set.

According to the triangle equality, we have $dist(o_i.\mu, o_j.\mu) > dist(o_i.\mu, Q.\mu) - dist(o_j.\mu, Q.\mu)$. Since $dist(o_i.\mu, Q.\mu) - dist(o_j.\mu, Q.\mu) > \tau$, we have $dist(o_i.\mu, o_j.\mu) > \tau$. For any object o_l that will be examined in the following steps, we have $dist(o_l.\mu, Q.\mu) > dist(o_i.\mu, Q.\mu)$ because the algorithm examines objects from near to far. Hence, we have $dist(o_l.\mu, Q.\mu) - dist(o_j.\mu, Q.\mu) > dist(o_i.\mu, Q.\mu) - dist(o_j.\mu, Q.\mu) > \tau$, which suggests that it is not possible that $temp$ can "grow" to a bigger candidate set with any unexamined object while satisfying the diameter bound τ. This completes the proof. ■

An example that exemplifies Lemma 1 is shown in Figure 1. Assume that $temp = \{o_j, o'_j\}$ is a candidate set in Map and o_i is being examined (note that o_i is further away than o_j and o'_j from Q). Since $dist(o_i.\mu, o_j.\mu) > \tau$, the union of $temp$ and another further-away and unexamined object, e.g., o_l, cannot become a new candidate set because o_l is definitely located further than the diameter bound τ from o_j. Thus, $temp$ can be removed safely.

We proceed to present BA in Algorithm 1. Assuming that we have a circular window with diameter τ, the baseline algorithm simulates a process where the circular window moves around in $\mathbb{R}^2$. In particular, the center of the window starts moving from the nearest object with respect to the query location $Q.\mu$, and it moves to the next nearest objects in the following iterations. When the circular window moves to a new location and covers different objects compared to those of its previous location, candidate sets are enumerated, and their scores are evaluated. Finally, the object sets with the top-k largest scores are returned.

Algorithm 1: Baseline Algorithm (BA)

Input : objects $\mathcal{O}$ and a query Q.
Output: The top-k results.

```
1  begin
2      Min-priority Queue: U(score, objectset) ← null;
3      Buffer Map ← ∅;
4      while o_i ← nextNearestNeighbor(Q.μ, O) do
5          Map ← Map ∪ {{o_i}};
6          if ξ({o_i}, Q) > U.GetScore(k) then
7              U.Enqueue(ξ({o_i}, Q), {o_i});
           /* Generating new candidate sets
              according to Lemma 1          */
8          foreach temp in Map do
9              if ∀o_j ∈
               temp, dist(o_i.μ, Q.μ) − dist(o_j.μ, Q.μ) ≤ τ
               then
10                 temp' ← temp ∪ o_i;
11                 Map ← Map ∪ temp';
12                 if ξ(temp', Q) > U.GetScore(k) then
13                     U.Enqueue(ξ(temp', Q), temp');
14             else
15                 Map ← Map \ temp;
16     return U.Top(k);
```

A min-priority queue U of size k is initialized to record the candidate object sets along with their scores (line 2). A memory buffer Map is created to keep all the set candidates for enumeration (line 3). The algorithm iteratively examines objects from near to far with respect to the query location $Q.\mu$ (line 4). When the algorithm examines object o_i, a candidate object set $\{o_i\}$ that only contains the object is added to Map (line 5); and if the score of the object set $\xi(\{o_i\}, Q)$ exceeds the minimum score in U, the score along with the object set $(\xi(\{o_i\}, Q), \{o_i\})$ is enqueued into U (lines 6–7).

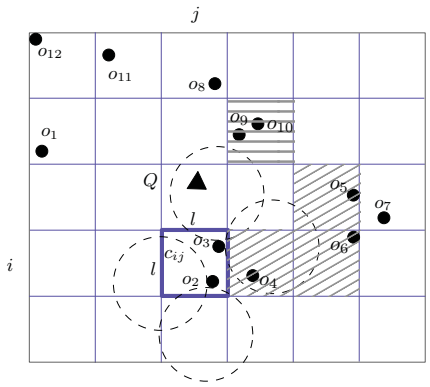


Fig. 2. An Indexing Example.

Further, if the distance of every object in a candidate set $temp \in Map$ to o_i is smaller than τ, the union of $temp$ and o_i becomes a new candidate set $temp'$. Later, $temp'$ is added to buffer Map, and is enqueued into U if its score exceeds the minimum score in U (lines 9–13). If there exists at least an object o_j further than τ from o_i, $temp$ can be safely removed from Map due to Lemma 1 (line 15). Finally, U that stores the top-k most relevant object sets is returned (line 16).

Theorem 2: Algorithm 1 reports correct exact results of the localitySearch query.

Proof: We prove the theorem by contradiction. Assume that Algorithm 1 misses a top-k object set $R' = \{o'_i, \ldots, o'_j\}$ in the results of a localitySearch query Q. This means that $\xi(R', Q)$ exceeds the minimum score in U but that there is not such an object set R' in Map. This contradicts the algorithm since Map keeps all possible object sets with pairwise distances smaller than τ. ■

B. Enhanced Algorithm

The exact enhanced algorithm (EEA) applies two performance-enhancing techniques to the BA algorithm. The first employs a grid index on the objects and applies approximate similarity bounds to prune irrelevant objects in indexing blocks. The second technique combines the algorithm with the derived distance bounds.

1) Approximating Candidate Bounds: We assume that all online objects $\mathcal{O}$ are indexed. In particular, a uniform grid is used. However, the proposed technique is also applicable with minor modifications to other indexing structures such as the R-tree.

Figure 2 shows several objects indexed by a grid. Cells that do not contain any object are not considered in our algorithm. The general idea of the first technique is to use similarity bounds to prune grid cells that can only contain irrelevant objects (i.e., the objects that cannot be in a result candidate). In order to prune a grid cell c_{ij}, we estimate the upper bound of a candidate set that can be found with the objects in c_{ij}. Let l be the grid cell size (we assume that $l < \tau$). The objects, whose distances from c_{ij} are within the diameter bound τ, are contained by cells in a square with side length $2\lceil \tau/l \rceil + 1$ centered at c_{ij}. To reduce the size of the candidates, we only consider cells in the square whose distances to the objects in c_{ij} are smaller than diameter τ. Consider Figure 2: assuming that $\lceil \tau/l \rceil = 2$ ($2\lceil \tau/l \rceil + 1$=5), it is possible for objects o_2 and o_3 in cell c_{ij} to combine with other objects in their 2-level neighbor cells (shaded in Figure 2) to become a candidate object set. In the figure, the shaded region does not include the cells that are further than τ away from objects in c_{ij} and those cells without any objects. Thus, we can examine a cell c_{ij} as well as its neighbors within a circular window with diameter τ to determine whether we can prune c_{ij}.

To speed up the examination, we use the following definitions and lemmas to approximate the ranking score of a candidate object set by estimating upper and lower bounds.

Definition 2: ($MIN\xi$) Given a set of cells $C = \bigcup c_{ij}$, we denote the set of objects that locate in C by $\mathcal{O}_C$. The minimal ranking score between an object set located in C and a query Q, denoted by $MIN\xi(C, Q)$, is defined as $MIN\xi(C, Q) = \min_{R \in 2^{\mathcal{O}_C}} \xi(R, Q)$, where $2^{\mathcal{O}_C}$ is the power set of $\mathcal{O}_C$.

Note that the computation of $MIN\xi(C, Q)$ is quite expensive especially when C contains many objects. We thus have to find an effective way to estimate the minimal ranking score. As we know each function in $\xi(R, Q)$ outputs positive values, e.g., $\Theta(R, Q) > 0$ where $R, Q \neq \emptyset$, we have $MIN\xi(C, Q) = \min_{R \in 2^{\mathcal{O}_C}} \xi(R, Q) > \min_{R \in 2^{\mathcal{O}_C}} (\alpha \cdot \Theta(R, Q)) + \min_{R \in 2^{\mathcal{O}_C}} (\beta \cdot \Im(R)) + \min_{R \in 2^{\mathcal{O}_C}} ((1 - \alpha - \beta) \cdot \Omega(R, Q))$. Thus, the estimated minimal score can be regarded as the summation of the minimal values of the three functions, which are much easier to compute since the minimal value of a function can be estimated locally by its properties, e.g. linearity.

We define $MAX\xi(C, Q)$ as the maximal ranking score between the objects in a region C and Q. According to the definitions of $MIN\xi(C, Q)$ and $MAX\xi(C, Q)$, we deduce the following score bounds for a candidate object set in a cell.

Lemma 2: Given a cell c_{ij}, and the set of objects $\mathcal{O}_{c_{ij}}$ in c_{ij}, the score of an object set $R \in 2^{\mathcal{O}_{c_{ij}}}$ in the cell c_{ij} satisfies the bounds as follows:

$$\underbrace{1 - \mathbb{D}(c_{ij}, Q.\mu)}_{min\ of\ \Theta} + \underbrace{\min(\Omega(R, Q.\psi))}_{min\ of\ \Omega} + \underbrace{1 - \sqrt{2}l/\tau}_{min\ of\ \Im} < \xi(R, Q) < \underbrace{1 - \mathcal{D}(c_{ij}, Q.\mu)}_{max\ of\ \Theta} + \underbrace{\max(\Omega(R, Q.\psi))}_{max\ of\ \Omega} + \underbrace{1}_{max\ of\ \Im}, \quad (5)$$

where $\mathbb{D}(\mathcal{O}_{c_{ij}}, Q.\mu)$ and $\mathcal{D}(\mathcal{O}_{c_{ij}}, Q.\mu)$ are the normalized maximal and minimal spatial proximity between c_{ij} and the query location $Q.\mu$. For simplicity, we omit the weight parameters α and β.

The proof is omitted due to the space limitation. To compute the bounds for a cell c_{ij} and its neighbors when there is a candidate set where objects locate both in c_{ij} and its neighbors, we develop the bounds shown in Lemma 3.

Lemma 3: Given a cell c_{ij} and its neighbors $\mathfrak{N}(c_{ij})$, the bounds of a candidate set R with objects both in c_{ij} and $\mathfrak{N}(c_{ij})$ are

$$\underbrace{1 - \mathbb{D}(\mathfrak{N}(c_{ij}) \cup c_{ij}, Q.\mu)}_{min\ of\ \Theta} + \underbrace{\min(\Omega(R, Q.\psi))}_{min\ of\ \Omega} + \underbrace{1 - 2\sqrt{2}l/\tau}_{min\ of\ \Im} < \xi(R, Q) < \underbrace{1 - \mathcal{D}(\mathfrak{N}(c_{ij}) \cup c_{ij}, Q.\mu)}_{max\ of\ \Theta} + \underbrace{\max(\Omega(R, Q.\psi))}_{max\ of\ \Omega} + \underbrace{1}_{max\ of\ \Im}.$$

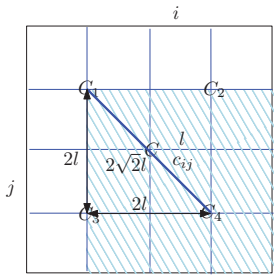


Fig. 3. Maximal Mutual Distance.

To obtain Lemma 3, the spatial proximities and the textual relevance are calculated with respect to a set of cells. Details are omitted due to the space limitation. Note that since there is at least one object in c_{ij} to form such a candidate set, we need to find the most and least relevant objects in the c_{ij} and $\mathfrak{N}(c_{ij})$, respectively, to get the bounds on textual relevance. For the part on mutual discrepancy, we use Figure 3 to illustrate the maximal mutual distance for such a candidate set, which is applied for lower bound computation. The shaded region in Figure 3 denotes a cell c_{ij} and its neighbors $\mathfrak{N}(c_{ij})$. The maximal distance between an object in c_{ij} and an object in $\mathfrak{N}(c_{ij})$ is $2\sqrt{2}l$. Based on Equation 3, we have that the minimal mutual average discrepancy is $1 - 2\sqrt{2}l/\tau$.

With Lemmas 2 and 3, we can prune some cells that cannot contain any candidate set before we exhaustively search the objects. For example, an idea similar to the threshold algorithm is: if a cell's upper bounds in the two lemmas are smaller than the k-th candidate score that has been seen, the cell can be pruned since one cannot find a candidate set in the cell or with objects in its neighbors to form a better candidate.

We find upper and lower bounds for each of the three terms used in the ranking function; and the sums of the upper and lower bounds are used as the final bounds. And we assume the three terms are independent. The bounds thus may not be the tightest ones but they are easy to compute, especially with the help of grid cells.

2) Bounding Search Space: The first technique estimates the approximate bounds of grid cells, which enables the algorithm to prune grid cells and not examine their objects. Next, we present the second technique based on bounded search regions. We proceed to discuss the related definitions and lemmas.

Definition 3: **Search region (R^2_{search}):** A search region R^2_{search} is a subset of $\mathbb{R}^2$ with objects that may form a top-k result.

Lemma 4: Given an arbitrary object o_i and the k-th candidate set $\hat{R_k}$, we have $o_i \in R^2_{search}$ if o_i satisfies: $dist(o_i.\mu, Q.\mu) \leq \Gamma_{t_i}$ where $\Gamma_{t_i} = (2 - (\xi(\hat{R_k}, Q) - 1)/\alpha) \cdot dist_{max}$.

Proof: Let R denote a candidate set satisfying $\xi(R, Q) > \xi(\hat{R_k}, Q)$, and let o_i be an arbitrary object in R. Without loss of generality, we assume that o_i is the object closest to the query location $Q.\mu$ among the objects in R. By definition, we have

$$\xi(R, Q) = \alpha \cdot (1 - \frac{dist(o_i.\mu, Q.\mu)}{dist_{max}}) + \beta \cdot \Omega(R, Q)$$
$$+(1 - \alpha - \beta) \cdot \Im(R) < \alpha \cdot (1 - \frac{dist(o_i.\mu, Q.\mu)}{dist_{max}}) + 1 - \alpha.$$

As we know $\xi(R, Q) > \xi(\hat{R_k}, Q)$, it then follows that

$$\alpha \cdot (1 - \frac{dist(o_i.\mu, Q.\mu)}{dist_{max}}) + 1 - \alpha \geq \xi(\hat{R_k}, Q).$$

We therefore obtain the bound on the search region as

$$dist(o_i.\mu, Q.\mu) \leq (2 - (\xi(\hat{R_k}, Q) - 1)/\alpha) \cdot dist_{max} = \Gamma_{t_i}.$$

This completes the proof. ■

Definition 4: **Suspect region ($R^2_{suspect}$):** A suspect region $R^2_{suspect}$ is a subset of $\mathbb{R}^2$, where subsets of the contained objects cannot form a top-k result set independently, but a subset may be contained by a top-k result set R, i.e., $\exists o_i, o_j \in R$ that satisfy $o_i \in R^2_{suspect}$ and $o_j \notin R^2_{suspect}$

We establish Lemma 5 to tell if an arbitrary object is in $R^2_{suspect}$.

Lemma 5: Given an arbitrary object o_i, we have $o_i \in R^2_{search}$ if o_i satisfies $\Gamma_{t_i} + \tau > dist(o_i.\mu, Q.\mu) > \Gamma_{t_i}$

Proof: The lemma is based on Lemma 4, where we have the bound of search the region as Γ_{t_i}. For a candidate R, if we have $o_j \in R$ where $o_j \in R^2_{search}$, we have $o_j \leq \Gamma_{t_i}$. By the diameter threshold, if $o_i \in R$, $o_i \notin R^2_{search}$, and $dist(o_i, o_j) < \tau$, then we have $\Gamma_{t_i} + \tau > dist(o_i.\mu, Q.\mu) > \Gamma_{t_i}$. ■

By now, we conclude that only the objects in the search region and the suspect region need to be examined. All other objects can be safely disregarded.

Example 1: Figure 4 shows an example to illustrate the bounded search space, where the query position is in the middle of the grid. During query processing, we obtain the boundary of search region Γ_{t_i} by Lemma 4. Further, we divide the grid region into three regions, R^2_{search}, the circular region, $R^2_{suspect}$, the shaded region, and R^2_{pruned}, the remaining region. We only need to search the candidate sets in R^2_{search} and $R^2_{suspect}$. Objects in $R^2_{suspect}$ have to be combined with ones in R^2_{search} to form candidate sets. Objects in R^2_{pruned}, e.g., o_4 and o_5, can be safely disregarded. Note that the first technique can be utilized before accessing objects in R^2_{search} and $R^2_{suspect}$.

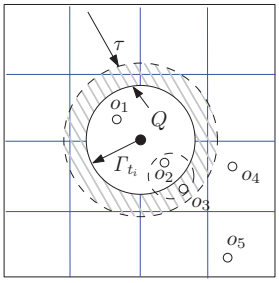


Fig. 4. Illustration of Bounded Search Space.

3) Algorithm Details: The enhanced algorithm EEA that uses the two proposed techniques is presented in Algorithm 2. The algorithm iterates on grid cells instead of iterating on individual spatial web objects (as Algorithm 1 does). The upper bound of the candidate object sets generated by a grid cell c_{ij} is calculated (line 5). The algorithm continues processing the subset $C_{c_{ij}}$ if its upper bound is greater than that of the minimum score in U (line 6). Otherwise, c_{ij} can be removed

Algorithm 2: Enhanced Exact Algorithm (EEA)

Input : Indexed spatial web objects $\mathcal{O}$ and a query Q.
Output: The top-k results.

```
1  begin
2      Min-priority Queue: U(score, objectSet) ← null;
3      Buffer: isEnumerated ← ∅;
4      while c_ij ← nextNearestGridCell(Q.μ, O) do
5          Estimate the upper bound of a candidate in the
           region C_cij centered at c_ij within τ;
6          if MAXξ(C_cij, Q) > ξ(R̂_k, Q) then
7              U ← GenerateSets(C_cij, U, isEnumerated)
8              isEnumerated ← isEnumerated ∪ C_cij;
9          else
10             CellSet.remove(c_ij);
11     return U.Top(k);
```

(line 10). When a region $C_{c_{ij}}$ possibly contains a candidate object set with a higher score, procedure $GenerateSets$, which can generate candidate object sets in the region, is called (line 7). While processing a region, it considers all its neighbor cells within the diameter bound that overlaps other regions. Algorithm 2 keeps track of grid cells that have been processed (line 8). It completes when there is no available c_{ij} (line 11).

Algorithm 3: Generate Set Candidate (GenerateSets)

Input : Set of grid cells $C_{c_{ij}}$, min-priority queue U, and buffer $isEnumerated$.
Output: updated U.

```
1  begin
2      foreach c'_ij ∈ C_cij do
3          if c'_ij ∉ isEnumerated then
4              calculate bounds of the single grid cell c'_ij;
5              if MAXξ(c'_ij, Q) < ξ(R̂_k, Q)||
               MAXξ(c'_ij, Q) < max_{c∈C_cij/c'_ij} (MINξ(c, Q))
               then flag the grid cell c'_ij;
6                   expand candidates with c'_ij else
               enumerate candidates in c'_ij;
7                   expand the candidates
8          else
9              flag the grid cell c'_ij;
10             expand candidates with c'_ij;
11             enumerate c'_ij with previously enumerated
               cells that are not in the same region;
12         for each R of candidates, call Update(R);
13     return the min-priority queue U;
14     Update(R) {
15         U ∪ R if ξ(R, Q) > ξ(R̂_k, Q);
16         update R²_pruned, R²_suspect, and R²_search;
17         prune objects in O;}
```

The procedure $GenerateSets$ is described in Algorithm 3. It exploits bounds of the cells in the buffer $isEnumerated$ to prune candidates in R^2_{search} according to the cell-based pruning technique. Thus, the procedure can generate candidates without performing an exhaustive search.

The procedure iterates through the cells intersecting with a region (i.e., a cell set) $C_{c_{ij}}$ (line 2 in Algorithm 3). In each iteration, it checks whether cell c'_{ij} was already processed or whether it is not possible to find a candidate object set in the cell. Once we find a candidate, we call **Update()** in line 12 in Algorithm 3 to enqueue the candidate such that we can get a better score $\xi(\hat{R}_k, Q)$. Meanwhile the procedure recalculates the regions in order to further prune candidates in subsequent iterations. Thus, c_{ij} in each iteration contains at least one object in R^2_{search} (line 4 in Algorithm 2). Otherwise, c_{ij} is pruned. In a c_{ij}, the objects in R^2_{pruned}, but not in $R^2_{suspect}$ are also pruned according to Lemma 5 (line 17 in Algorithm 3).

In an iteration, if a cell c'_{ij} has been processed, c'_{ij} is flagged (line 9 in Algorithm 3). A flagged cell indicates that there is no need to generate candidates that are only contained in the cell. The flagging is implemented to avoid duplicate computation. But an existing candidate may include some objects of c'_{ij} and may become a new candidate with a higher score. Thus the current candidates are expanded with the objects contained in c'_{ij} (line 6 in Algorithm 3).

For the overlapping cells that have been processed, we have to examine new candidates among these cells if they never co-appeared in the same $C_{c_{ij}}$. To efficiently implement this examination, we keep track of information in line 8 in Algorithm 2 such that we can know which co-appearing cells that have been enumerated.

C. Approximate Algorithms

In this section, we present a generic approximation algorithm that can use different greedy strategies. Based on the terms in Equation 4, systematic greedy strategies are exemplified and embedded into the generic approximate algorithm. The algorithms presented in Section III-C2–III-C4 are object-based and do not consider grid cells. The algorithm presented in Section III-C5 is grid based and searches grid cells instead of objects. The pseudo code for the specific algorithms is omitted due to the space limitation.

1) A Generic Approximate Algorithm: The generic approximate algorithm works iteratively. In each iteration, the algorithm forms an initial candidate set with the unvisited nearest neighbor o_i of query Q. An object is visited if we have already formed an initial candidate set with the object. With an initial candidate set, the generic algorithm tries to find objects within diameter bound τ to form a set with o_i using a greedy function f_t. We employ the pruning techniques developed in Section III-B.

The pseudocode of the generic algorithm is presented in Algorithm 4. Given an unvisited nearest neighbor o_i that is not in the pruned region ($o_i \notin R^2_{pruned}$), an initial candidate set R_{init} is created (line 4) for object o_i. This set is expanded by continuously including neighbor objects whose distances to o_i are within the diameter bound τ (lines 5–9).

A neighbor object is identified according to objective function f_t (line 9). We apply a procedure $CallObjective(f_t, Neighbors)$ to select a neighbor by f_t. If

the initial set together with the identified object results in a higher-score set, the identified object is added to the initial set. Otherwise, the expansion is completed, and we employ the second technique from Section III-B to prune the search region (line 9). The boundary of the pruned region is recalculated once a new candidate set is inserted into the result priority queue by procedure **Recalc()** in line 9. We omit the description of this procedure since it is quite similar to **Update()** in Algorithm 3. According to the space-based pruning technique (Section III-B2), the generic algorithm terminates when no unvisited objects can form an initial region in the search region (line 3). The top-k regions are reported finally in line 10.

The generic algorithm examines objects from near to far. For each candidate set, the distance from the initial object to the query location is that from the set to the query location. Thus, our greedy strategies consider the other two terms, the mutual average discrepancy and the textual relevance, in order to include neighbors within the diameter bound of the initial object.

Algorithm 4: Generic Approximate Algorithm (Generic)

Input : Spatial web Objects $\mathcal{O}$ and a query Q.
Output: $topK(\mathcal{O}, Q)$.

```
1  begin
2      Min-priority Queue: U(score, region) ← null;
3      while o_i ← nextNearestNeighbor(Q.μ, O) where
       o_i ∈ R^2_search do
4          R_init ← {o_i};
5          Neighbors ← {o_j} where dist(o_i.μ, o_j.μ) < τ;
6          while Neighbors ≠ ∅ do
7              o_j ← CallObjective(f_t, Neighbors);
8              remove o_j from Neighbors;
9              if ξ(R_init ∪ o_j, Q) > ξ(R_init, Q) then
               R_init ← R_init ∪ o_j else Recalc() and break
10     return U.mathitTop(k);
```

2) Approximate Algorithm 1 (Appro1): Algorithm Appro1 applies a greedy strategy in relation to the mutual average discrepancy function $\Im(R)$. It performs a best-first search within the diameter bound in each iteration in order to include objects that increase the $\Im(R)$ value. Specifically, for each initial set R_{init} formed by o_i in line 3 in Algorithm 4, Appro1 considers objects within distance τ and tries to include the objects according to

$$f_t = \underset{o_j \in Neighbors}{\operatorname{argmax}} \ \Im(R_{init} \cup \{o_j\}), \tag{6}$$

which is called by procedure $CallObjective(f_t, Neighbors)$ it line 7 in Algorithm 4. In an iteration, the neighbor search terminates if the score of an objective candidate is not increased. The highest scored candidate set before termination is inserted into the top-k results if its score exceeds the k-th one in line 9 in Algorithm 4. The process iterates until there is no unvisited object in R^2_{search} to form an initial set.

3) Approximate Algorithm 2 (Appro2): Given a query Q, the idea of algorithm Appro2 is to perform a best-first search in relation to $\Omega(R, Q)$. Initially, the unvisited nearest neighbor o_i is selected to form an initial candidate set R_{init} (line 3 in Algorithm 4). Then, Appro2 visits objects within diameter bound τ and tries to include them according to

$$f_t = \underset{o_j \in Neighbors}{\operatorname{argmax}} \ \Omega(o_j, Q), \tag{7}$$

which is called by procedure $CallObjective(f_t, Neighbors)$ in line 7 in Aglorithm 4. If a best selected object in $Neighbors$ cannot result in a higher-score object set, we terminate the search and try to insert a candidate (line 9 in Algorithm 4). The process ends when there is no unvisited object in R^2_{search}, as for Appro1.

4) Approximate Algorithm 3 (Appro3): Algorithm Appro3 uses both terms $\Im(R)$ and $\Omega(R, Q)$ with weights to expand the initial set R_{init}. The objective function is defined as

$$f_t = \underset{o_j \in Neighbors}{\operatorname{argmax}} \ (\beta \cdot \Im(R_{init} \cup o_j) + \gamma \cdot \Omega(o_j, Q)), \tag{8}$$

where β and γ control the importance of the two terms.

5) Approximate Algorithm 4 (Appro4): Algorithm Appro4 exploits the grid index that we used to improve the efficiency of the BA algorithm. With the grid index, it is possible for Appro4 to rank and select grid cells at a coarse granularity instead of examining single spatial web objects one by one. Appro4 identifies a region of grid cells that contains a set of objects with high-ranking scores. However, Appro4 does not aim to find the largest subset of objects in the region; instead, it reports all the objects contained by the region.

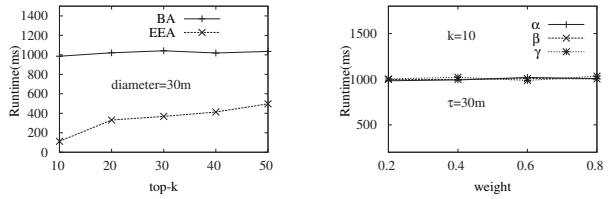
Given a query Q, Appro4 forms an initial set R_{init} with the unvisited nearest neighbor cell c_{ij} that contains at least one object $o_i \in R^2_{search}$. Since a cell may contain more than one object, we use the objective function in Equation 9 which takes both $\Im(R)$ and $\Omega(R, Q)$ into consideration.

$$f_t = \underset{c'_{ij} \in Neigbors}{\operatorname{argmax}} \ (\beta \cdot \Im(R_{init} \cup c'_{ij}) + \gamma \cdot \Omega(c'_{ij}, Q)), \tag{9}$$

where $\Omega(c'_{ij}, Q)$ is based on the normalization of the $\Omega(R, Q)$ term in Equation 2.

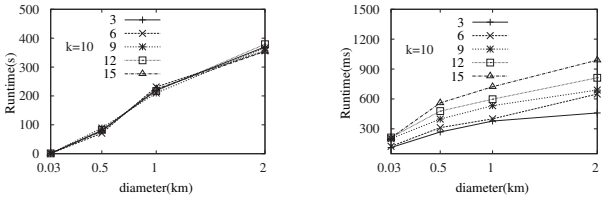
The objective function f_t uses both of the two normalized terms, $\Im(R)$ and $\Omega(R, Q)$, along with their weight parameters. The $\Im$ value of a cell is high if it contains a large number of objects. To simplify the computation without accessing the objects contained by a cell c_{ij}, we use the cardinality of a cell for $\Im(c_{ij})$ instead of exactly calculating the average pairwise distance. Thus we have $\Im(R_{init} \cup c'_{ij}) \approx |\mathcal{O}_{R_{init} \cup c_{ij}}|/(|R_{init} \cup c_{ij}| \cdot |\mathcal{O}|)$ where $|\mathcal{O}_{R_{init} \cup c_{ij}}|$ denotes the cardinality of objects in the expanded cell set $R_{init} \cup c_{ij}$ as defined in Definition 2, $|R_{init} \cup c_{ij}|$ is the number of cells, and $|\mathcal{O}|$ is the total number of objects.

We implement the grid cells with external index information on the objects, including the number of objects contained by a cell. As a result, to compute objective function f_t for a cell. It is not necessary to access the objects in the cell. For the objective function f_t, we can also use ideas similar to those presented in Sections III-C2 and III-C3. Since these ideas have similar functions and properties as the algorithms in Sections III-C2 and III-C3, we omit these heuristics. Note that we return a set R that consists of all the objects contained by the corresponding cell set that Appro4 finds in the top-k results. Although a subset of a result set R could be better than R according to their ranking scores, it is much easier to compute the approximate result set R.



(a) Varying k (b) Varying Weights (BA)

Fig. 6. Varying k and Weight Parameters for Exact Algorithms



(a) BA (b) EEA

Fig. 5. Varying Diameter Bound τ and Number of Keywords for Exact Algorithms

IV. Experimental Study

We employ three real-life datasets [4] in the performance study: **Hotel** has $20,790$ spatial web objects and $80,845$ keywords; **Web** has $579,727$ objects and $249,132,883$ keywords; **GN**, extracted from the U.S. Board on Geographic Names (geonames.usgs.gov), contains $1,868,821$ objects and $18,374,228$ keywords. Dataset Hotel is memory resident for algorithm evaluation, and the other two are used to evaluate the proposed algorithms when the data and index are disk resident.

We generate 5 query sets for each dataset. As the number of keywords in $Q.\psi$ may influence the text relevance, the 5 query sets have different numbers of query keywords, namely 3, 6, 9, 12, and 15. For each set, we randomly generate 50 locations. For each location, we first generate a set with 3 keywords by picking 3 keywords from the whole collection of keywords of a dataset to obtain a query. Then, we add sets of 3 keywords to incrementally generate the other four query sets. The procedure allows us to fix the location of a query while varying the number of keywords to study the effects. We report the average cost of 50 such queries for each query set.

We implement the algorithms in Java SE $1.7.0$ and run on an Intel(R) Core(TM) $i5$-2520M CPU @2.50GHz with 8GB RAM. Without further explanation, we set default values for weight parameters in the scoring function in Equation 4 as $\alpha = 0.3$ and $\beta = 0.3$. Each cell in the implemented grid index has a pointer to a keyword list that contains all the keywords associated with the contained objects. The cardinality of the contained objects in each cell is also recorded in the cell. Note that in this section, we define $\gamma = 1 - \alpha - \beta$.

Exact algorithms. The objective of this set of experiments is to study the effects on the performance of the exact algorithms of varying the settings of the parameters. We only use the small dataset Hotel to evaluate the exact algorithms as they are not scalable to large datasets. For the enhanced exact algorithm, we use a grid index with $100 \cdot 100$ cells by the default. The region containing all the objects in Hotel is thus divided into $100 \cdot 100$ cells. And diameter bound τ is 30m by default.

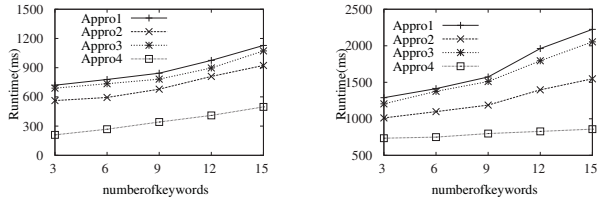
1) **Varying the number of keywords in $Q.\psi$ and diameter bound τ.** In this set of experiments, we run the 5 sets of queries containing 3–15 keywords and vary diameter bound τ. In the experiments, k is set to 10. The runtime of the enhanced exact algorithm (EEA) in Figure 5(b) is significantly shorter than that of the baseline algorithm (BA) in Figure 5(a). As we use diameter bound τ to release memory in BA, the number of candidates in memory grows with increasing τ according to Lemma 1. The runtime increases as τ increases, since more candidates are becoming memory residents, which lead to expensive enumeration. Figure 5(a) shows that BA is not sensitive to the number of query keywords. This is because BA has to enumerate all the combinations regardless of the text relevance score.

On the other hand, Figure 5(b) shows that the runtime of queries with more keywords is worse for EEA with a same diameter bound. According to the first technique of EEA, the reason is that more keywords enlarge the upper bound of a grid cell or a set of grid cells, which reduces the pruning capability of EEA. For a query set, the performance gets slightly worse as we increase the diameter in Figure 5(b). This finding fits with the nature of the second technique of EEA. Although a larger diameter means a larger suspect region, the performance depends on the number of contained objects in $\mathbb{S}^2_{suspect}$ as well as the number of sets that can be expanded with these objects.

2) **Varying the number of returned sets k.** Figure 6(a) shows the performance of the two exact algorithms when we vary the value of k that is the number of returned sets. The runtime of the exact baseline algorithm has no obvious change when we increase the value of k. However, the runtime of the enhanced exact algorithm increases with k. The reasons for these results are that the baseline algorithm only depends on τ, while the enhanced algorithm's bounds depend on the k-th best set of spatial web objects.

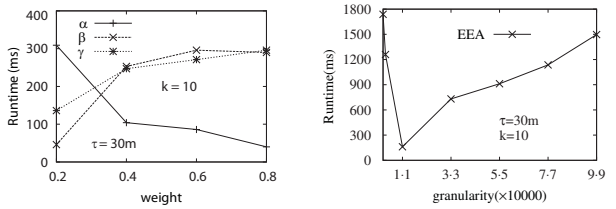
3) **Varying weight parameters α, β, and γ.** In this set of experiments, we vary one parameter from 0.2 to 0.8, and meanwhile keep the other two parameters constant at 0.3. As a result, the three parameters do not sum up to 1 in this set of experiments. The queries used have 3 keywords. Each line in Figure 6(b) denotes the runtime along with the variation of a parameter for BA. The findings suggest that the runtime of BA is not sensitive to the three parameters. But for EEA (see Figure 7(a)), the runtime gets better if we increase the value of α. The reason is that the increase of α means that EEA prefers sets close to the query point. This makes the first pruning technique more effective. On the other hand, the pruned region $\mathbb{S}^2_{pruned}$ at t_i decreases with the increase of β and γ, which consequently increases the runtime.

4) **Varying grid granularity.** The granularity is the total number of cells to index the objects. The findings are shown in Figure 7(b) where top-10 results ($k = 10$) with diameter bound 30m ($\tau = 30$) are returned. The results suggest that the $100 \cdot 100$ granularity is best for the default τ on Hotel, so we use a grid index with $100 \cdot 100$ cells as the default. The first two granularities used in the figure are $20 \cdot 20$ and $30 \cdot 30$. Note that the side length l of a grid cell should satisfy



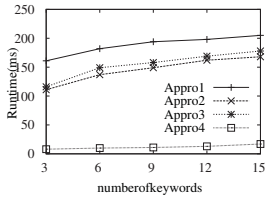
(a) Web (b) GN

Fig. 9. Varying Number of Keywords



(a) Varying Weights (EEA) (b) Varying Grid Granularity

Fig. 7. Varying Weight Parameters and Grid Granularity for EEA



(a) Hotel

Alg.	top-1	top-5	top-10
Appro1	0.99	0.99	0.98
Appro2	0.95	0.92	0.91
Appro3	0.99	0.98	0.97
Appro4	0.91	0.76	0.63

Varying k on Hotel

(b) Approximate Ratio on Hotel

Fig. 8. Keywords and Approximate Ratio on Hotel

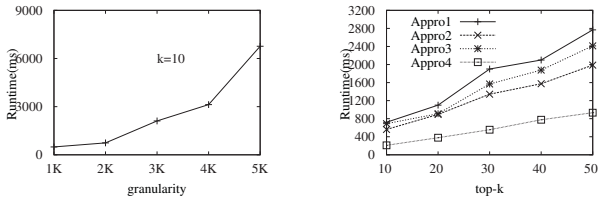
$l < \tau$, but it cannot be too small. For example, each grid cell at most contains one spatial web object when l is smaller than the minimal distance between two objects in the dataset, and thus the use of cells becomes ineffective.

Approximation algorithms. We run the four approximation algorithms on Hotel with the same default settings as for exact algorithms. To evaluate the computational efficiency and scalability on larger datasets, we use GN and Web for disk-based query processing and indexing. The default grid granularity for the approximation algorithms on the two large datasets is $1000 \cdot 1000$, k is 10, and the diameter is 30m by default.

1) **Varying the number of keywords.** Figure 8(a) shows the runtime of the four approximation algorithms on Hotel. The algorithms are evaluated using query sets with varying numbers of keywords and report top-10 results. The cell-based algorithm Appro4 performs the best. Figure 9 shows the scalability of our approximation algorithms on large datasets. The runtime of BA is not shown in Figure 9(a) as it cannot finish within 30 minutes. Regarding EEA, as the runtime superlinerly scales from 10^4 to 10^5 in Figure 9(a), we also remove it from the figure. For the same reason, we only present the runtime of the four approximation algorithms for the even larger dataset GN used Figure 9(b). Figure 9(a) and Figure 9(b) indicate that the four approximation algorithms scale well with the number of keywords. The results also show that Appro4 scales much better than the other three. This is because Appro1, Appro2, and Appro3 compare and enumerate individual spatial web objects. In contrast, Appro4 can avoid the enumeration and approximates the sets directly according to grid cells. Figure 9(b) shows that Appro2 performs the best among the three object-based methods since Appro1 and Appro3 have to calculate pairwise average distance in the objective function in order to select a containing object within its neighborhood. This costly step has to be executed whenever there is any change to the candidate set. For Appro2, on the other hand, text relevance of one object to a query Q does not change with the candidate set. Thus, it is not necessary for Appro2 to recompute the text relevance for one object if it is already computed for Q.

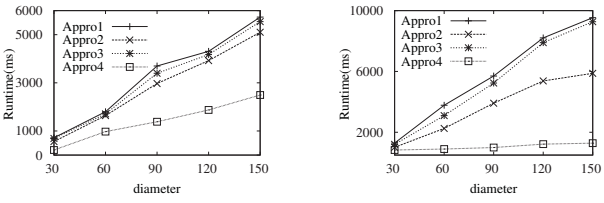
2) **Approximation ratio.** We evaluate the results returned by the four approximation algorithms in terms of the exact results. The evaluation is performed with 3 keyword query sets on Hotel. We compute the average score (s_1) of the top-k sets returned by EEA and then compute the average score (s_2) returned by the approximation algorithms. With the average scores, we obtain approximation ratios as s_2/s_1. The findings on Dataset Hotel are reported in Figure 8(b). On Dataset Web, the approximation ratios for Appro1, Appro2, Appro3, and Appro4 are 0.91, 0.84, 0.88, and 0.81 when top-1 results are returned. On Dataset GN, the approximation ratios for Appro1, Appro2, Appro3, and Appro4 are 0.88, 0.86, 0.87, and 0.75 when top-1 results are returned. The results thus show that the approximation algorithms achieve good performance in the setting.

3) **Varying parameters.** Figure 10(a) shows that the performance of Appro4, which employs the cell-based pruning technique, decreases when we increase the granularity from $1000 \cdot 1000$ to $5000 \cdot 5000$ (represented as 1K–5K in the figure). As Appro4 searches the regions by visiting grid cells, it has to enumerate more regions when there are more cells. For varying k and diameter bound τ on GN and Web, we observe findings in Figure 10(b), Figure 11(a), and Figure 11(b). In the figures, Appro4 performs the best among the four when we increase the number of results as well as the diameter bound.



(a) Varying Grid Granularity (Web) (b) Varying k on Web

Fig. 10. Varying Grid Granularity and Number of Returned Results k



(a) Varying τ on Web (b) Varying τ on GN

Fig. 11. Varying Diameter Bounds

V. Related Work

[Spatial Keyword Queries]. Web objects extracted from web pages have attracted much attention [1], [2], [9]. Spatial web objects, which possess geographical locations and text descriptions, are gaining in prevalence [3], and the problem of extracting locations from web pages (e.g., [1]) has been studied, yielding spatial web objects that can subsequently be queried [10], [11]. Most of the existing work on spatial keyword queries aims to find single objects that are close to a query location and are relevant to query keywords [10], [12]. Many recent studies are reviewed by Cao et al. [3].

[Collective Spatial Keyword Queries]. Two studies [5], [1] find a single group of objects. The query in the studies takes a set of m keywords as an argument and returns m objects of minimum diameter that match the m keywords. They do not consider a query location. Several studies propose to find groups of spatial web objects according to a query location. They consider a collective spatial keyword query [4], [6] that takes as arguments a location and a set of keywords. Cao et al. [4] present two types of collective spatial keyword queries. The first type of query finds a group of spatial web objects that collectively cover the query keywords and whose total distances to the query location are minimized. The second type of query finds a group of objects such that they collectively cover the keywords and the sum of their diameter and their maximum distance to the query point is minimized. The collective spatial keyword query is substantially different from our problem. It considers keywords as Boolean conditions and finds a group of objects that collectively covers all the query keywords. It only considers two specific distance-based measurements in ranking the groups of objects. In contrast, we retrieve the k most relevant sets with a general and unified approach, and the semantics are different from those in collective spatial keyword querying. For example, we include text relevance when ranking sets of objects.

[Continuous Queries]. There exist studies on continuous queries [13], [14]. These queries report results with the change of query location. To optimize the computation, safe zones are used. The query we consider in this paper is a one-time query rather than a continuous query. We leave a continuous version for future work.

[Scoring Functions]. The general problem of maximizing a scoring function has received much attention in both theoretical and practical settings. Some studies [15], [16], [17], [18] present several approximate methods to contend with particular functions. Most of the functions have good properties, e.g., submodularity, supermodularity, monotonity, additivity, and linearity. Such properties make it possible to approximate the objective functions with tight bounds. In contrast, our objective function does not have such properties, but our solutions are easy to compute especially with the help of grid cells.

VI. Conclusion and Future Work

We propose the problem of localitySearch that returns the best top-k sets of co-located spatial web objects that are ranked by both spatial proximity and textual relevance. This problem is NP-hard. We provide efficient exact algorithms and a generic approximate algorithm which is scalable for large datasets. The generic approximate algorithm is able to use a variety of greedy strategies. The experimental findings demonstrate the efficiency of the solutions as well as effectiveness properties.

localitySearch points to direction in spatial querying where a spatial query incorporates non-spatial preferences, and many other scenarios, e.g., in social-spatial search, can be considered as an extension of this work. We leave more efficient and continuous solutions for future work.

Acknowledgments This research was supported in part by the Geocrowd Initial Training Network, funded by the European Commission as an FP7 Peoples Marie Curie Action under grant agreement number 264994. Siyuan Liu was supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and the Pinnacle Lab at Singapore Management University.

References

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

[2] L. R. A. Derczynski, B. Yang, and C. S. Jensen, "Towards context-aware search and analysis on social media data," in *EDBT*, 2013, pp. 137–142.

[3] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *ER*, 2012, pp. 16–29.

[4] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD*, 2011, pp. 373–384.

[5] 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.

[6] 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.

[7] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[8] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[9] Q. Qu, J. Qiu, C. Sun, and Y. Wang, "Graph-based knowledge representation model and pattern retrieval," in *FSKD*, 2008, pp. 541–545.

[10] 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.

[11] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 656–665.

[12] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou, "Approximate string search in spatial databases," in *ICDE*, 2010, pp. 545–556.

[13] 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, pp. 7:1–7:47, 2013.

[14] 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.

[15] M. A. Bender, D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, H. Meijer, and C. A. Phillips, "Communication-aware processor allocation for supercomputers: Finding point sets of small average distance," *Algorithmica*, vol. 50, no. 2, pp. 279–298, 2008.

[16] A. Borodin, H. C. Lee, and Y. Ye, "Max-sum diversification, monotone submodular functions and dynamic updates," in *PODS*, 2012, pp. 155–166.

[17] B. Chandra and M. M. Halldórsson, "Approximation algorithms for dispersion problems," *J. Algorithms*, vol. 38, no. 2, pp. 438–465, 2001.

[18] M. Drosou and E. Pitoura, "Search result diversification," in *SIGMOD*, 2010, pp. 41–47.