Retrieving Top-k Prestige-Based Relevant Spatial Web Objects

Xin Cao[†] Gao Cong[†] Christian S. Jensen[‡]

[†]School of Computer Engineering, Nanyang Technological University, Singapore
xcao1@e.ntu.edu.sg, gaocong@ntu.edu.sg

[‡]Department of Computer Science, Aarhus University, Denmark
csj@cs.au.dk

ABSTRACT

The location-aware keyword query returns ranked objects that are near a query location and that have textual descriptions that match query keywords. This query occurs inherently in many types of mobile and traditional web services and applications, e.g., Yellow Pages and Maps services. Previous work considers the potential results of such a query as being independent when ranking them. However, a relevant result object with nearby objects that are also relevant to the query is likely to be preferable over a relevant object without relevant nearby objects.

The paper proposes the concept of prestige-based relevance to capture both the textual relevance of an object to a query and the effects of nearby objects. Based on this, a new type of query, the Location-aware top-k Prestige-based Text retrieval (L*k*PT) query, is proposed that retrieves the top-k spatial web objects ranked according to both prestige-based relevance and location proximity.

We propose two algorithms that compute L*k*PT queries. Empirical studies with real-world spatial data demonstrate that L*k*PT queries are more effective in retrieving web objects than a previous approach that does not consider the effects of nearby objects; and they show that the proposed algorithms are scalable and outperform a baseline approach significantly.

1. INTRODUCTION

Studies suggest that at least some 20% of all web queries have local intent, meaning that the queries target local content. In step with the web being used increasingly by mobile users, this percentage can be expected to increase. Next, geo-positioning is increasingly available for mobile devices, e.g., by means of built-in GPS receivers. This enables web users who query for local content to provide their locations to services. Search engines already recognize local intent, and specialized services, e.g., maps and yellow-page services, that target local content continue to proliferate. For example, travel sites such as TripAdvisor and TravellersPoint offer services that enable users to find hotels with particular facilities and located in particular regions.

Several proposals already exist for the querying for geo-located web content, termed spatial web objects. A *location-aware keyword query* takes a location and specified keywords as arguments and returns web objects that are ranked according to both spatial proximity and text relevance relative to the query. Some proposals [10, 21] view keywords as Boolean predicates, filtering out web objects that do not contain the keywords and ranking the remaining objects based on their spatial proximity to the query. Other proposals [7, 8] combine spatial proximity and textual relevance using a linear ranking function.

Existing work treats objects as independent when ranking them for a given query. However, spatial web objects are not independent. A relevant object whose nearby objects are also relevant to the query is preferable when compared to a relevant object without relevant nearby objects. One reason is that if the object a user chooses to visit does not work out then there are other nearby relevant objects. Another is that a user may intend to visit several objects (e.g., to compare prices). For example, a user may prefer to visit a location with many restaurants or shops instead of a location with only one restaurant or shop.

A preference for clusters of relevant objects may explain the phenomenon that similar businesses tend to co-locate. For example, car dealerships tend to co-locate. We speculate that they benefit from the spatial proximity: together, they attract more customers to such an extent that this compensates for the increased competition.

It is the objective of this paper to support this phenomenon in spatial web search. This is done by developing a notion of object "prestige" that takes into account the presence of nearby objects that are also relevant to a query. This notion of prestige is then used for the ranking of the query results.

We believe that this is the first study on supporting this inter-object relationship in location-aware keyword querying. However, in non-spatial web search, inter-document relationships have been exploited to improve effectiveness of document retrieval. For example, a PageRank-like algorithm is applied to the document similarity graph (built based on document similarity rather than web links), thus significantly improving the effectiveness of document retrieval [18] and question answers [9].

A further benefit of supporting this notion of prestige is that even if the description of an object does not contain the query terms, the object can still be identified as relevant. This occurs if the object has a text description that matches those of nearby objects that in turn contain the query terms. For example, consider a query for "spring roll" and two close objects with descriptions "best Chinese restaurant in Boston" and "Chinese restaurant offering spring rolls." The two descriptions are similar, and the latter contains the query term. So although the first object's description does not contain the query term, the object is identified as relevant to the query.

Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

To illustrate the effect of supporting prestige-based object relevance, consider the query "shoes" at location P in Figure 1. Circles represent shops selling shoes or jeans, with centers representing locations and areas representing relevance to the query. Existing spatial keyword search techniques (e.g., [8]) rank R5 as the top-1 result since R5 is relevant and closest to P. However, R4 is more attractive because it has more nearby shops that are relevant to the query and also is close to the query location.

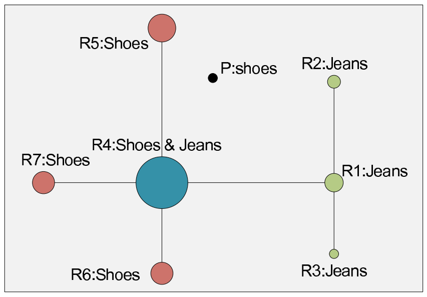

Figure 1: Prestige propagation example

In our proposal for prestige-based relevance (denoted as PR) of objects to queries, the PR score of an object is affected by the PR scores of its neighbors. This motivates us to employ a PageRank-like random walk mechanism for the propagation of prestige.

Conceptually, given the example query above, the PR scores of the objects are computed as follows: We build a graph with the objects as the nodes. Two nodes are connected by an edge if the objects are close and their text descriptions are similar. Myriads of random surfers are initially placed at the nodes that contain the query term "shoes" (i.e., R4, R5, R6, R7). The number of random surfers at a node is proportional to the textual relevance between the node and the query, which is the initial prestige-based relevance of the node. At each step, each random surfer either moves to an adjacent node following a link in the graph with a certain walking probability (depending on the distance between the nodes), or it randomly jumps to the initial set of nodes containing "shoes" without following any link, again with a certain probability (depending on the well-known damping factor [5]). The expected percentage of surfers at each node eventually converges, and the converged percentage of surfers at a node represents the PR score of the node.

The concept of PR is inspired by the concept of personalized PageRank [16], where a subset of web pages share the initial prestige uniformly (rather than all web pages as in PageRank), and its applications to keyword search in Entity-Relation graphs [1, 6].

The above random walk process has unique features that render a direct application of PageRank [5] inadequate. PageRank is used to compute the objects' global importance, which is query-independent, and it has no preferences for any particular nodes. Our problem is very different: each query has a set of preference objects based on the initial relevance scores, and thus random surfers start from this set of objects and jump to them. For example, given a "jeans" query at point P, R1 is the best result rather than R4, which is best if the query is for "shoes."

We propose a new type of query, called the *Location-aware top-k Prestige-based Text retrieval* (LkPT) query, that takes into account both location proximity and prestige-based text relevance (PR). The query retrieves a list of k objects ranked according to their spatial distances and PR scores with respect to the query.

The LkPT query is expensive to compute, especially due to the PR scores. A straightforward approach to computing the LkPT query is to adapt the algorithms for computing Personalized PageRank Vectors (PPVs) [1, 4, 6, 12, 16] to computing the PR scores of all objects in the spatial object graph, to use an R-tree for computing spatial distances, and then to combine the two scores. However, this solution is expensive. First, it is expensive to compute PR scores for a large graph at query time using existing algorithms for PPVs. Second, it is impractical to pre-compute PPVs for each node in terms of either pre-computation time or the storage requirements [16]—$|V|^2$ space is needed to store the PPVs for all objects. Third, it is a waste to compute the spatial distance and PR scores of all objects and then rank them to find the top-k results.

We note that the spatial object graph has unique properties that render it different from the web link graph [16] and the entity-relation graph [1, 6].

- Similar spatial objects often co-locate geographically. For example, shops often co-locate, as do, e.g., bars. Therefore, spatial objects tend to naturally form subgraphs.
- The number of nodes in a subgraph is constrained by geography. Thus, subgraphs will be of relatively modest size.

Properties like these enable us to develop approaches that speed up PR scoring. We propose two novel algorithms for the efficient computation of the LkPT query.

ES-EBC (Early stop extended bookmark coloring): We prove that if the distance between a node and the query point exceeds a certain threshold, the node will not affect the PR scoring of the top-k objects. Therefore, we need only consider nearby nodes when propagating PR, which speeds up the PR scoring substantially. We also show how to estimate lower and upper bounds on the PR score of each object in each iteration during scoring. Utilizing these bounds, we derive conditions for when further iterations will not change the ranking order of the top-k objects and we stop iterating.

S-EBC (Subgraph-based extended bookmark coloring): We propose an approximate solution to PR scoring with performance guarantees. We organize spatial objects and their text descriptions using the external memory IR-tree [8]. Hence, the spatial objects are grouped into subgraphs based on their locations, with each subgraph corresponding to a leaf node of the IR-tree. We prove that the PR scores of the nodes in a subgraph can be computed by PR propagation within the subgraph and contributions from the border objects that connect the subgraph with other subgraphs. This enables PR scoring w.r.t. subgraphs rather than on the whole graph. Next, we propose a novel approach to estimating an upper bound on the PR scores of the objects in each subgraph. This bound together with the distance of a subgraph to the query is used to choose which subgraphs to process and in which order.

Empirical studies with real data offer insight into the effectiveness of LkPT queries and the efficiency of the proposed algorithms.

The rest of the paper is organized as follows. Section 2 defines the LkPT query. Section 3 details the proposed algorithms. In Section 4, we report on the empirical studies. Section 5 concludes, and an appendix offers a variety of additional detail.

2. PROBLEM DEFINITION

Let $\mathcal{D}$ be a set of spatial objects. Each object o in $\mathcal{D}$ has a text description $o.\psi$ and a location $o.\mu$. Similarly, a *Location-aware top-k Prestige-based Text retrieval* (LkPT) query $Q = \langle\psi, \mu\rangle$ has a location $Q.\mu$ and a set of keywords $Q.\psi$.

Let $\mathrm{Dist}(Q, o)$ denote the Euclidean distance between the locations of query Q and object o, and let $\mathrm{Sim}(Q, o)$ denote the relevance between the keyword component of query Q and the text description of object o. We use the Vector Space Model [23], one of the most popular ranking functions, for computing $\mathrm{Sim}(Q, o)$, while using the TF-IDF weighting scheme to represent the text descriptions of objects (details are in Appendix A).

To compute the PR scores of objects, we define a weighted, undirected graph $\mathcal{G} = (V, E)$ over $\mathcal{D}$, where each node in V corresponds to a spatial object and edge set E includes an edge $\langle o_i, o_j \rangle$ iff the following two conditions are satisfied: 1) $\mathrm{Dist}(o_i, o_j) \leq \lambda$ and 2) $\mathrm{Sim}(o_i, o_j) \geq \xi$, where λ and ξ are threshold parameters. The weight of edge $\langle o_i, o_j \rangle$ in E is $\mathrm{Dist}(o_i, o_j)$.

Prestige-Based Relevance (PR). The final PR vector $\vec{p}$ fulfills the following equation:

$$\vec{p} = (1-\alpha)\mathbf{C}^{\mathbf{T}}\vec{p} + \alpha\vec{u_Q},$$

$$\vec{u_Q} = [v_1, ..., v_{|\mathcal{D}|}]^T, v_i = \mathrm{Sim}(Q, o_i), 1 \leq i \leq |\mathcal{D}|, \quad (1)$$

where $\mathbf{C}$ is the normalized adjacency matrix of graph $\mathcal{G}$ such that $\sum_{j \in V} C(b,j) = 1$, where $C(b,j)$ represents the normalized weight from node b to node j; and column vector $\vec{u_Q}$ is the initial PR vector in which each element is the relevance of an object.

Parameter α represents the probability of a random surfer jumping to the set of initially relevant spatial objects ($v_i > 0$) instead of following the edges in the graph. Interestingly, parameter α can be used to balance the relevance of an object and the effect of its relevant neighbors, i.e., the parameter allows for tuning according to user-specific requirements. In particular, smaller values of α favor objects with nearby relevant objects, while larger values of α favor objects with high initial PR scores.

To understand the random walk process for each object b, its PR score $\vec{p}(b)$ can be rewritten as follows:

$$\vec{p}(b) = \alpha\vec{u_Q}(b) + (1-\alpha)\sum_{j \in V} C(j,b)\vec{p}(j), \quad (2)$$

where $\vec{u_Q}(b)$ is the initial PR score of b.

The iterative computation diffuses the PR score of each object across the graph. In the beginning, each object gets its initial PR score according to its text relevance to the query. At each step, an α fraction of the PR score is held by each node, while the remaining $(1-\alpha)$ flows by following the links of the graph. This propagation continues until all the prestige is distributed cross the graph. The final PR scores take into account both the original relevance scores and the effect of neighbor nodes.

The PR vector is inspired by the of personalized PageRank vector (PPV) [16] of preference vector $\vec{u_Q}$. In the original PPV [16], a set of preferred objects in the preference vector are assigned uniform initial scores, while we assign an initial score to an object that is in proportion to its text relevance to the query.

L*k*PT Query Definition. Intuitively, an L*k*PT query retrieves k objects from database $\mathcal{D}$ ranked according to a combination of their distances to the query location and their PR scores for the query. Formally, given a query $Q = (\psi, \mu)$, where $Q.\psi$ is a location descriptor and $Q.\mu$ is a set of keywords, the objects returned are ranked according to a ranking function $f(\mathrm{Dist}(Q, o), \Pr(Q, o))$, where $\mathrm{Dist}(Q, o)$ is the Euclidian distance between Q and o and $\Pr(Q, o)$ is the PR score of o with respect to Q. An L*k*PT query becomes an L*k*T query [8] in the extreme case of $\alpha = 1$ (Equation 2), i.e., we disregard the effects of nearby relevant objects.

Problem Statement. We address the problem of efficiently answering L*k*PT queries.

The paper's proposals are applicable to a wide range of ranking functions that are monotone with respect to the distance proximity $\mathrm{Dist}(Q, o)$ and the PR score $\Pr(Q, o)$. We follow existing work [8, 20] and use a linear combination of the normalized factors for ranking an object o with respect to a query Q:

$$\mathrm{RS}(Q, o) = (1-\beta)(1-\Pr(Q, o)) + \beta\frac{\mathrm{Dist}(Q, o)}{maxD} \quad (3)$$

where $\beta \in (0, 1)$ is used to balance the PR score and the location proximity; Euclidian distance $\mathrm{Dist}(Q, o)$ between query Q and object o is normalized to a value between 0 and 1 by a constant $maxD$, which can be the maximum distance between two objects in $\mathcal{D}$ or the maximum distance that can be accepted by the users; and $\Pr(Q, o)$ is the PR score of object o w.r.t. query Q and usually takes a value between 0 and 1. This function computes the ranking score of each object given an L*k*PT query.

Note that parameter β allows to set the preference between the PR score and the location proximity at query time.

3. PROPOSED SOLUTIONS

3.1 Baseline Algorithm

As a baseline, we present an improvement of the straightforward solution mentioned in the introduction.

In the straightforward solution, we compute PR scores for all objects and the distances between all objects and the query, upon which we rank the objects based on the combined scores. We focus on computing the costly PR scores.

We choose to adapt the bookmark-coloring algorithm (BCA) [4], an elegant algorithm for computing PPVs, to computing the PR scores. We first extend the in-memory BCA algorithm to work in secondary memory. We read the graph in large blocks that each exploit the memory available, and do the iterative propagation in a per-block manner. The computation stops when the termination condition for the graph is met. A block is likely to be read and written multiple times since it may receive PR scores from other blocks that need to be distributed. Second, we BCA, which works on unweighed graph for a single preferred object, to support PR score computation for a general preference vector $\vec{u_Q}$ on a weighted graph.

Algorithm 1 details the resulting Extended BCA (EBC) algorithm. Let $\vec{p}$ denote the PR score that each object already has, let $\vec{q}$ denote the PR score that each object needs to distribute, and let $outPR$ be the vector of the sum of the PR scores that need to be distributed in each graph block.

Algorithm 1 EBC(Q)

```
Input: query $Q$
Output: The PR score of each object
 1: compute the text relevance of each object to $Q$ and compute $\vec{u_Q}$
 2: $\vec{q} \leftarrow \vec{u_Q}, \vec{p} \leftarrow \vec{0}, outPR \leftarrow \vec{0}$
 3: $blockQueue \leftarrow$ NewPriorityQueue()
 4: for each object $b$ do $outPR(b.block) \leftarrow outPR(b.block) + \vec{q}(b)$
 5: for each block $bg$ do $blockQueue$.Enqueue($bg$)
 6: while $\|\vec{q}\|_1 \geq \varepsilon$ do
 7:     $bg_i$ = $blockQueue$.Dequeue()
 8:     read the graph block $bg_i$
 9:     $outPR(bg_i) \leftarrow 0$
10:     $Queue \leftarrow$ NewQueue()
11:     for each object $n$ s.t. $n \in bg_i$ and $\vec{q}(n) > 0$ do
12:         $Queue$.Enqueue($n$)
13:     while not $Queue$.Empty() do
14:         $b \leftarrow Queue$.Dequeue()
15:         if $\vec{q}(b) > \epsilon$ then
16:             $\vec{p}(b) \leftarrow \vec{p}(b) + \alpha\vec{q}(b)$
17:             for each out-neighbor $j$ of $b$ do
18:                 if $\vec{q}(j)$ = 0 and $j \in bg_i$ then $Queue$.Enqueue($j$)
19:                 $outV \leftarrow (1-\alpha)C(b,j)\vec{q}(b)$
20:                 $\vec{q}(j) \leftarrow \vec{q}(j) + outV$
21:                 if $j \notin bg_i$ then
22:                     $outPR(j.block) \leftarrow outPR(j.block) + outV$
23:             $\vec{q}(b) \leftarrow 0$
24:     $blockQueue$.Update()
25: return $\vec{p}$
```

We compute the text relevance of each object to the query Q using an inverted list index and then construct the preference vector $\vec{u_Q}$ according to Equation 1 (line 1). We use a priority queue $blockQueue$ whose key is the accumulated outgoing PR score that needs to be propagated in each block (lines 4–5). In each graph block, we modify the propagation mechanism of BCA to accommodate edge weights and multiple objects in the preference vector.

We use a queue $Queue$ to store the objects that have PR scores that need to be distributed (lines 10–12). Specifically, for the PR $\vec{q}(b)$ that needs to be distributed at an object b, we assign $\alpha\vec{q}(b)$ to b (line 16) and $(1-\alpha)\vec{q}(b)$ to its neighbors according to the edge weights (lines 17–20). We update the PR score that needs to be distributed for each block (lines 21–22).

When the PR score of each object that needs to be distributed is smaller than the propagation threshold ϵ, we stop the propagation within a block (line 15). When the PR score to be distributed is smaller than the tolerance threshold ε (line 6), we stop the propagation over the graph and return PR vector $\vec{p}$, each element of which represents the PR score $\Pr(Q, o)$ of object o. We thus use ε and ϵ as the termination conditions, as in BCA [4] and its variant [13].

This method is inefficient because it computes PR scores and distances for all objects. The PR scores of objects are inter-dependent and need to be computed together, while the distance scores can be computed individually. Thus, inspired by the Threshold algorithm [11], we develop two improved baseline algorithms that avoid unnecessary distance score computations. These two algorithms are covered in Appendix B.1.

3.2 Early Stop EBC Algorithm (ES-EBC)

PR scores are much more costly to compute than distances because they require iterations over possibly large graphs. We thus proceed to propose two new techniques for speeding up the computation of PR scores. First, we show how to stop the iterative PR score computation early, using a new stopping condition (Theorem 1). Second, we show how to disregard objects further away from the query than a certain distance (Theorem 2). All proofs are found in Appendix C.

As before, let $\vec{p}$ denote the vector of the PR score that each object already has, and let $\vec{q}$ denote the vector of the outgoing PR score that each object needs to distribute.

LEMMA 1. *Let $\vec{p}_i(b)$ denote the PR score of object b in the i-th iteration during PR scoring. Then $\vec{p}_i(b) \leq \vec{p}_{i+1}(b)$.*

LEMMA 2. *Given an object b, the final PR value $\vec{\hat{p}}(b)$ of b and the value $\vec{p}_i(b)$ of b in the i-th iteration fulfill the following:*

$$\vec{p}_i(b) \leq \vec{\hat{p}}(b) \leq \vec{p}_i(b) + \alpha^2 max(\vec{q}_i) + (1-\alpha)\|\vec{q}_i\|_1,$$

where $max(\vec{q}_i)$ is the maximum element in $\vec{q}_i$ in the i-th iteration, and $\|\cdot\|_1$ is the 1-norm.

Lemma 2 follows previous work [13]. The current ranking score $\text{CRS}(b)$ of an object b is computed according to Equation 3 at the current (i-th) iteration of PR scoring. We estimate lower and upper bounds on the ranking score for each object in the i-th iteration as follows:

$$\begin{aligned} &\text{Upper}(b) = \text{CRS}(b) \\ &\text{Lower}(b) = \text{CRS}(b) - (1-\beta)(\alpha^2 max(\vec{q}_i) + (1-\alpha)\|\vec{q}_i\|_1) \end{aligned} \tag{4}$$

The upper bound holds because the distance between an object and the query is constant, while its PR score increases in each iteration (Lemma 1). We obtain the lower bound based on Lemma 2 and Equation 3.

THEOREM 1. *Let priority queue L_r record the current top-$(k+1)$ objects seen, the key being the objects' PR scores. It is guaranteed that the top-k objects have been found if the k-th and the $(k+1)$-st objects, represented by $L_r(k)$ and $L_r(k+1)$, respectively, satisfy the following condition:*

$$\text{Upper}(L_r(k)) < \text{Lower}(L_r(k+1))$$

The PR score computation stops iterating when the condition in Theorem 1 is satisfied, i.e., the current top-k objects all have smaller final ranking scores than those of all other nodes.

THEOREM 2. *Given a query Q, a set of candidate objects C, and a spatial cell Ω_i, objects contained in Ω_i can be disregarded during propagation if the following is satisfied:*

$$\text{minDist}(Q, \Omega_i) > \lambda \log_{1-\alpha} \frac{\epsilon}{outC(\Omega_i)} + \text{Dist}(Q, o_s),$$

where $\text{minDist}(Q, \Omega_i)$ is the minimum distance between Q and Ω_i, λ is the distance threshold used when building the object graph; α and ϵ are as explained in Algorithm 1; o_s is the object in C furthest away from query Q; and $outC(\Omega_i)$ stores the aggregated PR score that needs to be distributed in Ω_i.

The condition stated in Theorem 2 guarantees that the objects in cell Ω_i will neither become top-k results nor affect the PR scores of the top-k results. Thus, the objects in the cell can be disregarded during the propagation of scores.

The algorithm that exploits the early stopping conditions first divides the graph into blocks according to the locations of the spatial objects such that each block fits into memory. Each block is further divided into a grid of spatial cells. Then nearest neighbors are retrieved incrementally [14] using the R*-tree [3]. For each nearest neighbor object o, the block graph containing o is read cell by cell: for each cell the algorithm checks whether it can be pruned according to Theorem 2; if it cannot, it reads the part of the graph corresponding to the cell. Then it iterates in the block to get the local PR scores for the objects in the block.

The algorithm keeps track of the current top-$(k+1)$ objects. When the ranking score of the k-th object is smaller than the lower bound (the minimum possible) ranking score of the current nearest-neighbor object o, i.e., $\Delta = \beta\frac{Dist(Q,L_r(k))}{maxD}$), the nearest-neighbor retrieval stops because no unseen object has a lower ranking score than has object o (since unseen objects no closer to the query than o) and thus cannot be a top-k object.

This way, we obtain a set of candidate top-k objects. However, these do not necessary constitute the final result since PR scores were only propagated inside a block. We need to propagate the PR scores across blocks while still using Theorems 1 and 2. Additional explanations and pseudo-code are available in Appendix B.2.

3.3 Subgraph-Based EBC Algorithm (S-EBC)

3.3.1 Overview of the Algorithm

Recall that the baseline and ES-EBC algorithms need to propagate PR scores on the whole graph. To compute PR scores within some selected subgraphs, we develop several techniques.

First, we show that PR scores can be computed by the combination of two parts: the PR score propagation within a subgraph and the PR contributed by the propagations from other subgraphs, which can be computed from pre-computed distribution vectors of the border nodes that connect the subgraph with other subgraphs (see Section 3.3.2). This enables us to compute PR scores with respect to a subgraph.

Second, we propose an approach to identifying the subgraphs that need to be checked to find the top-k results, thus avoiding checking all subgraphs. This is enabled by a novel approach to estimating upper bound PR scores of objects in a subgraph, given a query.

Specifically, we organize the spatial objects by extending the external memory IR-tree [8]. The spatial objects are grouped into subgraphs so that each subgraph corresponds to a leaf node of the IR-tree. We enrich the nodes of the tree with pre-computed information (see Section 3.3.3) and show that by utilizing this information, we can compute an upper bound PR score at each node for a query.

The upper bound PR scores together with the distance of a subgraph to the query are used to choose which subgraphs to process at query time. If the best estimated ranking score of nodes in a subgraph exceeds (the smaller a score, the better) the score of the k-th object, the subgraph cannot contribute to the top-k results and can be pruned.

Based on this, we propose an approximate algorithm with performance guarantees for answering L*k*PT queries (see Section 3.3.4). The approximation occurs because we do not process all subgraphs.

3.3.2 Subgraph-Based PR Scoring

We present a decomposition method that enables us to compute PR scores with regard to subgraphs.

Assume that we have already partitioned the graph $\mathcal{G}$ into m subgraphs $\mathcal{G}_1, ..., \mathcal{G}_m$ (to be discussed in Section 3.3.3). Let $border(\mathcal{G}_i)$ be the set of border objects of $\mathcal{G}_i$ that connect $\mathcal{G}_i$ with other subgraphs. Also, let H be the set of the border objects of all subgraphs, i.e., $H = \bigcup_{i\in[1,m]} border(\mathcal{G}_i)$.

For each border object b, we pre-compute and store a vector $\vec{\mathrm{GP}}_b$ that describes how to distribute the unit initial PR score from b over the whole graph. Note that the number of border objects is much smaller than the number of objects in the database. In a vector $\vec{\mathrm{Pr}}_b$, most of the elements are 0 since in spatial graphs, a node b usually only affects the objects in nearby subgraphs. We do not need to store the value 0.

We proceed to show that the PR score vector of an object, which is assigned the unit initial PR score, can be computed by propagation within its subgraph together with the propagations contributed by the border nodes, which we capture in pre-computed PR score vectors of border nodes. We denote the PR scores computed within a subgraph as the local PR scores ($\vec{\mathrm{LP}}$), and we denote the PR scores on the whole graph as the (global) PR scores ($\vec{\mathrm{Pr}}$). We have the following lemma and theorem:

LEMMA 3. *Given a node b and a subgraph $\mathcal{G}_i$ containing object b, we can compute the PR score vector of b as follows:*

$$\vec{\mathrm{Pr}}_b = \mathrm{L}\vec{\mathrm{Pr}}_b + \sum_{h\in border(\mathcal{G}_i)} \vec{\mathrm{AP}}_b(h) \cdot \vec{\mathrm{Pr}}_h,$$

where $\vec{\mathrm{AP}}_b(h)$ is the accumulated PR score of border node h during the local propagation within subgraph $\mathcal{G}_i$.

THEOREM 3. *Given a query Q, its PR score vector is computed as:*

$$\vec{\mathrm{Pr}}_Q = \sum_{j=1}^{m}\sum_{o\in\mathcal{G}_j} \mathrm{Sim}(Q,o)(\mathrm{L}\vec{\mathrm{Pr}}_o + \sum_{h\in border(\mathcal{G}_j)} \vec{\mathrm{AP}}_o(h)\cdot\vec{\mathrm{Pr}}_h)$$

$\mathrm{Sim}(Q, o)$ *is the similarity of query Q and the description of object o according to the vector space model (Appendix A).*

Theorem 3 allows us to decompose the computation of PR scoring. Given a query Q and a subgraph, we compute the text relevance to Q of each object in the subgraph. We distribute these scores following the links within the subgraph: when we reach a border node, the node accumulates the value distributed to it; if we meet a non-border node in the subgraph, we increase its PR score by a portion of the scores and distribute the rest to its out-neighbors, as in Algorithm 1. Having processed all subgraphs (i.e., we distribute PR scores within each subgraph), we use the global PR vector of each border object in a subgraph to update the PR scores for all the objects according to Theorem 3.

3.3.3 Indexing and Ranking Score Estimation

To efficiently process an L*k*PT query, we need to organize the spatial objects into subgraphs. We proceed to briefly introduce the index structure used for organizing objects and then focus on how to estimate an upper bound PR score for each node for a single-term as well as a multi-term query.

Index structure—IR-tree. We extend the IR-tree [8] index structure to organize spatial objects and capture the pre-computed information needed for upper bound estimation. It is also used to partition the graph.

A leaf node L in the IR-tree contains a number of entries of the form $\langle o, o.\mu\rangle$, where o is the identifier of an object and $o.\mu$ is the bounding rectangle of the object. A leaf node corresponds to a subgraph and contains a pointer to the subgraph at the node and the global PR score vector of the border objects of the subgraph.

A non-leaf node R contains a number of entries of the form $\langle cp, \Omega\rangle$, where cp points to a child node and Ω is the minimum bounding rectangle of all rectangles of entries in the child node.

Each node contains a pointer to an inverted file that describes the objects in the subtree rooted at the node. The inverted file for a node X contains: 1) A vocabulary of all distinct terms in the text descriptions of the objects in the subtree rooted at X. 2) A set of posting lists, each of which relates to a term t. Each posting list is a sequence of pairs $\langle cp, wt_{cp,t}\rangle$, where cp is a child of X and $wt_{cp,t}$ is the upper bound PR score of objects in the subtree rooted at cp for term t.

It is challenging to develop an effective approach to estimating the upper bound PR score of a node even for a single-keyword query (i.e., $wt_{cp,t}$), much less a multi-keyword query. It is computationally prohibitive to pre-compute the exact PPVs for each object [16], and this also holds for PR score vectors since they need similar computation. Note that the purpose of pre-computing and storing the upper bounds is that we can then utilize them to prune the search space at query time for an L*k*PT query (this will become clear shortly).

Upper bound PR score for single-keyword queries. We first consider a leaf node; it is straightforward to derive an upper bound for a non-leaf node from those of its child nodes. We estimate the upper bound for a leaf node by the sum of the upper bound PR scores from the propagation within the node and the maximum contribution from other subgraphs. Let L be a leaf node that corresponds to a subgraph $\mathcal{G}_i$. Let $\mathrm{maxGPr}(t, L)$ denote the estimated upper bound PR score of the objects in L for (query) term t. To estimate $\mathrm{maxGPr}(t, L)$, we need the initial PR score of a subgraph $\mathcal{G}_i$ for a query term t, denoted as $\mathrm{IPS}(t, \mathcal{G}_i)$.

DEFINITION 1. *The initial PR score of $\mathcal{G}_i$ for t is computed as follows:*

$$\mathrm{IPS}(t,\mathcal{G}_i) = \sum_{o\in\mathcal{G}_i} \mathrm{Sim}(t,o)$$

Here, $\mathrm{Sim}(t, o)$ *is the similarity between term t and the description of object o according to the Vector Space Model (Appendix A).*

We first present a lemma on how to estimate a maximum local PR score within a subgraph $\mathcal{G}_i$ given a query term t. This is the PR score without considering the effects of other subgraphs.

LEMMA 4. *The maximum local PR score* $\mathrm{maxLPr}(t, \mathcal{G}_i)$ *can be estimated as:*

$$\text{maxLPr}(t, \mathcal{G}_i) = \frac{1+\alpha-\alpha^2}{2-\alpha} \max_{o\in\mathcal{G}_i}(\text{Sim}(t,o)) + \frac{1-\alpha}{2-\alpha}\text{IPS}(t,\mathcal{G}_i)$$

Here, $\max_{o\in\mathcal{G}_i}(\text{Sim}(t,o))$ *is the largest initial PR score in subgraph* $\mathcal{G}_i$ *for term t.*

Based on this lemma, we get the following theorem.

THEOREM 4. *We estimate* $\text{maxGPr}(t, \mathcal{G}_i)$*, the global upper bound PR score of an object in* $\mathcal{G}_i$ *for t, as follows:*

$$\text{maxGPr}(t, \mathcal{G}_i) = \text{maxLPr}(t, \mathcal{G}_i) + \sum_{j\neq i} \text{IPS}(t, \mathcal{G}_j) \cdot \max_{bn\in border(\mathcal{G}_j), b\in\mathcal{G}_i} (\vec{\text{Pr}}_{bn}(b))$$

$\text{IPS}(t, \mathcal{G}_i)$ *is computed according to Definition 1, and* $\text{IPS}(t, \mathcal{G}_j)\cdot \max_{bn\in border(\mathcal{G}_j), b\in\mathcal{G}_i}(\vec{\text{Pr}}_{bn}(b))$ *represents the maximum possible PR score that can be propagated from* $\mathcal{G}_j$ *to a node in* $\mathcal{G}_i$*.*

We can now explain the weight $wt_{cp,t}$ in the inverted file. At a leaf node X, the upper bound of each object cp is computed as $\text{Sim}(cp.\psi, t)$ (defined in Appendix A); At the parent node X of a leaf node, $wt_{cp,t} = \text{maxGPr}(t, G_{cp})$ (computed according to Theorem 4). For other nodes, $wt_{cp,t}$ is the largest PR score among the child nodes of cp, i.e., $\max_{R\in cp.children()} wt_{t,R}$.

Upper bound PR score for multi-keyword queries. Based on the pre-computed upper bound PR score for a single keyword query, we propose an approach to estimating the upper bound PR score for a multi-keyword query.

LEMMA 5. *Given a subgraph* $\mathcal{G}_i$*, we compute its initial PR score (*IPS*) for a query Q as follows:*

$$\text{IPS}(Q, \mathcal{G}_i) = \sum_{t\in Q.\psi \bigcap \mathcal{G}_i.\psi} \frac{w_{\mathbf{Q}.\psi,t}}{W_{\mathbf{Q}.\psi}} \text{IPS}(t, \mathcal{G}_i),$$

where $w_{Q.\psi,t}$ *and* $W_{\mathbf{Q}.\psi}$ *are defined in Appendix A and* $\text{IPS}(t, \mathcal{G}_i)$ *is computed according to Definition 1.*

Lemma 5 provides a way of computing the initial PR scores of a query Q in a subgraph $\mathcal{G}_i$. We proceed to present how to estimate the upper bound PR score of each node in the IR-tree for a query Q.

DEFINITION 2. *Given a query Q and a node X in an IR-tree, the largest possible PR score of objects in X,* $\text{maxPr}(Q, X)$*, is defined as:*

$$\text{maxPr}(Q, X) = \sum_{t\in Q.\psi \bigcap X.\psi} \frac{w_{Q.\psi,t}}{W_{\mathbf{Q}.\psi}} \text{maxGPr}(t, X),$$

where $w_{Q.\psi,t}$ *and* $W_{\mathbf{Q}.\psi}$ *are defined in Appendix A.*

THEOREM 5. *Given a query Q and a leaf node X that encloses a set of objects* $XO = \{o_1, \ldots, o_m\}$*, the following holds:*

$$\forall o \in XO; (\text{maxPr}(Q, X) \geq \text{Pr}(Q, o))$$

We proceed to present the *minimum spatial-PR score distance*, minRS, which is needed for the query processing. Given a query Q and a node X in the IR-tree, the metric minRS offers a lower bound on the actual spatial-PR score distance between query Q and the objects in node X. This bound can be used to order and efficiently prune the search space in the index.

DEFINITION 3. *Given a query Q and a node X, the minimum spatial-PR distance, denoted by* $\text{minRS}(Q, X)$*, is defined as:*

$$\text{minRS}(Q, X) = (1-\beta)(1-\text{maxPr}(Q, X)) + \beta\frac{\text{Dist}(Q.\mu, X.\Omega)}{maxD}$$

Here, $\text{maxPr}(Q, X)$ *is the upper bound PR score of objects in X for query Q (cf. Definition 2).*

THEOREM 6. *Given a query Q and a node X whose rectangle encloses a set of objects* $XO = \{o_1, \ldots, o_m\}$*, the following is true:*

$$\forall o \in XO; (\text{minRS}(Q, X) \leq \text{RS}(Q, o))$$

The extended IR-tree used in this paper and the original IR-tree [8] share a similar data structure. However, the inverted files in the two indexes store different contents. The novelty of the extended IR-tree is its approach to estimating upper bound PR scores.

3.3.4 Subgraph-Based EBC Algorithm (S-EBC)

We proceed to describe the S-EBC algorithm that exploits the techniques just presented.

The main idea is to choose subgraphs that are more likely to contain top-k results for a query Q and then compute the PR scores in the selected subgraphs. For a node X in the IR-tree, we estimate its largest possible PR score according to Definition 2, and we compute its distance to the query. Thus, we can compute the smallest possible ranking score ($\text{minRS}(X, Q)$ in Definition 3; the smaller the score, the better).

We use a priority queue $queue$ to keep track of the nodes that have yet to be visited; the smallest possible ranking score is used as the key. When the head of the queue is a leaf node, i.e., its corresponding subgraph $\mathcal{G}_i$ has the lowest possible ranking score, we process the subgraph using the approach from Section 3.3.2. When the propagation within the subgraph completes, we have a local PR score for each object in $\mathcal{G}_i$, and we use the PR scores held by the subgraph's border objects to update the global PR scores of all the objects (for object o, according to Theorem 3).

We proceed to process the next subgraph using the priority queue. The processing continues until the smallest possible ranking score of the unvisited head node of the priority queue exceeds the ranking score of the current k-th result; we can then stop since no unvisited object can become a top-k result.

It is guaranteed that the unprocessed subgraphs (leaf nodes) do not contain top-k objects. However, they may affect the PR scores of the current top-k objects. To ensure this effect is within a certain bound, some postprocessing is needed. When building an IR-tree, we append the following pre-computed information to each leaf node (subgraph $\mathcal{G}_i$) of the IR-tree: a set of the IDs of the subgraphs that affect $\mathcal{G}_i$, denoted by $\mathcal{G}_i.Near$ and a factor describing the maximum possible effect of a subgraph on an object in $\mathcal{G}_i$ (e.g., for a subgraph $\mathcal{G}_j$, the factor is $\max_{bn\in border(\mathcal{G}_j), b\in\mathcal{G}_i}(\vec{\text{Pr}}_{bn}(b))$, according to Theorem 4).

In the postprocessing, we find the set SS of subgraphs containing the current top-k objects. For each subgraph $\mathcal{G}_i$ in SS, we then find $\mathcal{G}_i.Near$, the set of subgraphs that affect the PR scores of objects in $\mathcal{G}_i$. For each subgraph $\mathcal{G}_j$ in $\mathcal{G}_i.Near$, if it is not yet processed, we compute its maximum possible effect on an object in $\mathcal{G}_i$, denoted by $\text{maxEF}(\mathcal{G}_j, \mathcal{G}_i)$, according to Theorem 7. We sort the subgraphs in $\mathcal{G}_i.Near$ in ascending order of $\text{maxEF}(\mathcal{G}_j, \mathcal{G}_i)$, and we then find the m-th subgraph for which $SumErr = \sum_{j=[1,m-1]} \text{maxEF}(\mathcal{G}_j, \mathcal{G}_i) < \sigma$ and $SumErr + \text{maxEF}(\mathcal{G}_m, \mathcal{G}_i) \geq \sigma$.

Then starting from the m-th subgraph, for each subgraph in the sorted $\mathcal{G}_i.Near$, we do local propagation and update the PR scores of objects. We then update the list of the current top-k objects. If new objects are in the top-k and their corresponding subgraphs are not in SS, we include these subgraphs in SS and repeat the above steps until we have processed all subgraphs in SS. The postprocessing ensures that the maximal possible error in the ranking score of each top-k object is smaller than the error bound σ.

THEOREM 7. *Given a query Q, the maximum possible PR score that subgraph $\mathcal{G}_j$ can propagate to an object in $\mathcal{G}_i$ is:*

$$\mathrm{maxEF}(\mathcal{G}_j, \mathcal{G}_i) = \mathrm{IPS}(Q, \mathcal{G}_j) \max_{bn \in border(\mathcal{G}_j), b \in \mathcal{G}_i} (\vec{\mathrm{Pr}}_{bn}(b))$$

Pseudo-code and further explanations are given in Appendix B.3.

4. EXPERIMENTAL STUDY

4.1 Experimental Settings

Algorithms. In addition to the two proposed algorithms, ES-EBC and S-EBC, we compare with the two baseline approaches in Appendix B.1. As Baseline 1 outperforms Baseline 2 significantly, we only report results for Baseline 1 and refer to this as "Baseline."

Data and queries. We use three datasets that are real or based on real datasets. Table 1 shows some properties ff the datasets; additional descriptions are provided in Appendix D. Hotel is a small dataset while GN is much bigger. The objects in both datasets have short descriptions. Web is a medium-sized dataset whose objects have long descriptions. We evaluate our approaches on these three different datasets.

Property	Hotel	Web	GN
Total number of objects	20,790	579,727	1,868,821
Total number of unique words	602	2,899,175	222,409
Total number of words	80,845	249,132,883	18,374,228

Table 1: Dataset properties

We generate 4 query sets, in which the number of keywords is 1, 2, 3, and 4, respectively, in the space of GN, and we generate 4 similar query sets for the space of Spam and Hotel. Each set comprises 200 queries, and each query is randomly generated. We report average costs of the queries in each query set.

Setup. The IR-tree index structure is disk resident, and the page size is 16KB. The number of children of a node in the IR-tree is computed given the fact that each node occupies a page. This translates to 400 children per node in our implementation. The default values for parameters are as follows: k is 10, the number of query keywords is 2, α is 0.5 (Equation 2), and β is 0.5 (Equation 3) for all algorithms. S-EBC needs an extra parameter σ (to control its error bound; Section 3.3.4) that is set to 0.0001. Two threshold parameters for building graphs λ and ξ (Section 2) are set at 2 km and 0.5, respectively.

All algorithms were implemented in VC++, and run on an Intel(R) Core(TM)2 Duo CPU T7500 @2.66GHz with 2GB RAM.

4.2 Experimental Results

The reported results are on GN if not stated otherwise.

Varying k in LkPT. Figure 2 show the results of varying k when using the default settings for the other parameters.. Note that the y-axis uses a logarithmic scale.

We can see that ES-EBC and S-EBC significantly outperform (by an order of magnitude) the baseline for all values of k. ES-EBC performs better than the baseline due to the early stopping of propagation and punning of cells during score propagation. S-EBC outperforms the other two methods since it computes PR scores w.r.t. selected subgraphs rather than the whole graph as do the other two methods. As expected, the runtimes of all the approaches increase slightly with increasing k.

Varying the number of keywords. Figure 3 shows that ES-EBC and S-EBC outperform the baseline for different numbers of keywords. All algorithms need more time as the number of keywords increases since they need to process more words.

Varying α. Figure 4 shows that ES-EBC and S-EBC significantly outperform the baseline for all values of α. We also note that the

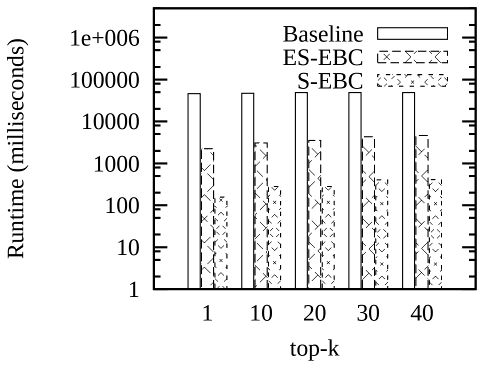

Figure 2: Varying k

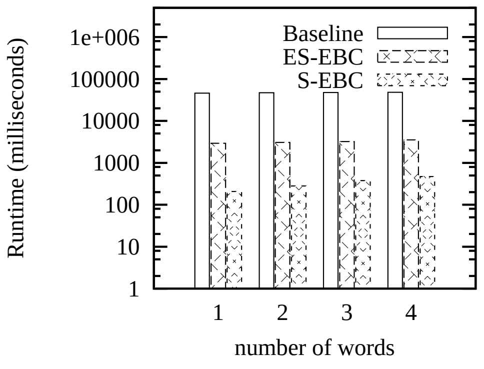

Figure 3: Varying # keywords

runtime increases as α decreases. This is because it takes longer for the propagation to converge with a smaller α.

Recall that parameter α can balance the relevance of an object versus the effect of its relevant neighbors. In particular, smaller values of α favor nodes with nearby relevant nodes, while larger values of α favor nodes with high initial PR scores. At one extreme, when $\alpha = 1$, the LkPT query is essentially the same as LkT query [8] that does not consider the effect of nearby relevant objects. Hence, we can see the overhead of considering the inter-relationships between objects by comparing with the runtime at $\alpha = 1$

Varying β. Figure 5 shows the results. Parameter β in Equation 3 allows users to set their preferences between the PR score and spatial proximity. A large β means that the spatial distance is more important, while a small β means that the PR score is more important.

As expected, ES-EBC and S-EBC perform better for larger β—they benefit from spatial proximity being given higher weight. When spatial proximity is given very low weight, ES-EBC nearly cannot prune any cell, and S-EBC nearly cannot prune any subgraph and needs process the entire IR-tree.

Scalability. To evaluate scalability, we generate 5 datasets containing from 2 to 10 million data points: we generate new locations by copying the locations in GN to nearby locations while maintaining the real distribution of the objects; and for each new location, a document is selected at random from the text descriptions of the objects in GN. Figure 6 shows that ES-EBC and S-EBC scale linearly with the size of the dataset, but that Baseline does not scale.

Summary on other experiments. The following experiments are included in Appendix D: *Experiments on Web and Hotel.* The results on the two datasets are consistent with those on GN. *Varying the parameters for building graphs*. When we increase λ or re-

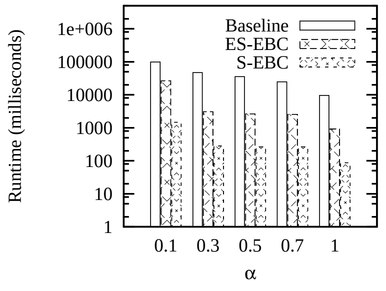

Figure 4: Varying α

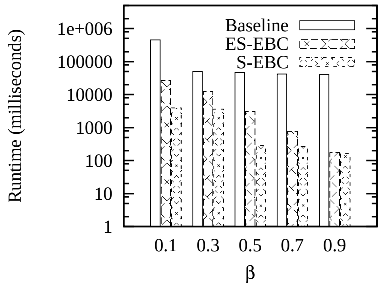

Figure 5: Varying β

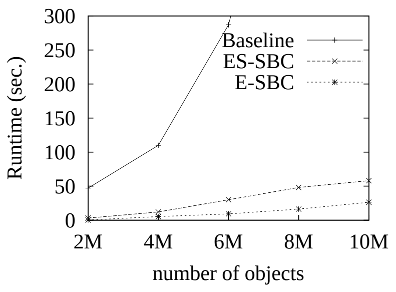

Figure 6: Scalability with data size

duce ξ (to generate denser graphs), the runtime of all algorithms increases since PR score propagation takes longer to complete. *Space requirements*. Baseline and ES-EBC use the same disk space, while S-EBC needs more space. *Effectiveness of LkPT queries*. Both ES-EBC and S-EBC are more effective than a previous method that does not consider prestige propagation.

5. RELATED WORK

We briefly cover the most closely related work; Appendix E offers additional coverage.

Spatial Keyword Search. Local search services, such as Google Maps and Yahoo! Local, allow the retrieval of local commercial content, e.g., relating to shops and restaurants, for a given query consisting of a location and a set of keywords. However, the algorithms used are not available. Similar applications include online yellow pages, online travel websites, and hotel search websites. Recent studies [7, 8, 10, 21, 22] on geographical retrieval address the problem of spatial keyword search. No existing work on spatial keyword search takes into account the inter-relationships among spatial objects.

Personalized PageRank. Jeh and Widom [16] propose the PPV concept and remark that pre-computing and storing all PPVs is impractical, as is computing PPVs at query time, since the computation of PPV needs an iterative computation over the web graph. Several algorithms [4, 12, 13, 16] have been proposed to compute the personalized PageRank vector (PPV). However, Langville and Meyer [19] write in a well-known survey "If the holy grail of real-time personalized search is ever to be realized, then drastic speed improvements must be made, perhaps by innovative new algorithms." PPVs are also used for keyword queries in entity-relation graphs [1, 6], as surveyed in Appendix E.

6. CONCLUSIONS

We propose a new type of query, the LkPT query, that retrieves the top-k spatial web objects ranked according to both location proximity and so-called prestige-based relevance that considers both the text relevance of an object to a query and the presence of nearby objects that are relevant to the query. We develop two baseline algorithms and propose two new algorithms to process the LkPT query. Results of empirical studies on real data demonstrate the effectiveness of LkPT the query and the efficiency of the new algorithms.

In future research, it is of interest to provide support for updates, as well as to consider the effect of nearby objects on rankings for other types of queries.

Acknowledgments

The research was conducted when the authors were employed at Aalborg University, Denmark. C. S. Jensen is an Adjunct Professor at University of Agder, Norway.

7. REFERENCES

[1] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: authority-based keyword search in databases. In *VLDB*, pp. 564–575, 2004.
[2] Z. Bar-Yossef and L.-T. Mashiach. Local approximation of pagerank and reverse PageRank. In *CIKM*, pp. 279–288, 2008.
[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pp. 322–331, 1990.
[4] P. Berkhin. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Math.*, 3(1):41–62, 2006.
[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7), 1998.
[6] S. Chakrabarti. Dynamic personalized PageRank in entity-relation graphs. In *www*, pp. 571–580, 2007.
[7] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pp. 277–288, 2006.
[8] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
[9] G. Cong, L. Wang, C.-Y. Lin, Y.-I. Song, and Y. Sun. Finding question-answer pairs from online forums. In *SIGIR*, pp. 467–474, 2008.
[10] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pp. 656–665, 2008.
[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
[12] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.*, 2(3):333–358, 2005.
[13] M. Gupta, A. Pathak, and S. Chakrabarti. Fast algorithms for top-k personalized PageRank queries. In *WWW*, pp. 1225–1226, 2008.
[14] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265–318, 1999.
[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM TOIS*, 20(4):422–446, 2002.
[16] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pp. 271–279, 2003.
[17] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Exploiting the block structure of theweb for computing PageRank. Stanford University Technical Report 2003-17.
[18] O. Kurland and L. Lee. Pagerank without hyperlinks: structural re-ranking using links induced by language models. In *SIGIR*, pp. 306–313, 2005.
[19] A. Langville and C. Meyer. Deeper inside PageRank. *Internet Math.*, 1(3):335–380, 2004.
[20] B. Martins, M. J. Silva, and L. Andrade. Indexing and ranking in Geo-IR systems. In *GIR*, pp. 31–34, 2005.
[21] 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*, pp. 688–699, 2009.
[22] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pp. 155–162, 2005.
[23] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comp. Surv.*, 38(2):6, 2006.

APPENDIX

A. VECTOR SPACE MODEL

The vector space model is defined as follows.

$$\mathrm{Sim}(Q.\psi, o.\psi) = \frac{\sum_{t \in \mathbf{Q}.\psi \bigcap \mathbf{o}.\psi} w_{\mathbf{Q}.\psi,t} w_{\mathbf{o}.\psi,t}}{W_{\mathbf{Q}.\psi} W_{\mathbf{o}.\psi}}, \text{ where}$$

$$w_{\mathbf{Q}.\psi,t} = \ln(1 + \frac{|D|}{f_t}),\ w_{\mathbf{o}.\psi,t} = 1 + \ln(tf_{t,\mathbf{o}.\psi}) \quad (5)$$

$$W_{\mathbf{Q}.\psi} = \sqrt{\sum_t w^2_{\mathbf{Q}.\psi,t}},\ W_{\mathbf{o}.\psi} = \sqrt{\sum_t w^2_{\mathbf{o}.\psi,t}}$$

Here f_t is the number of objects whose text descriptions contain the term t, and $tf_{t,\mathbf{o}.\psi}$ is the frequency of term t in $\mathbf{o}.\psi$. $w_{\mathbf{o}.\psi,t}$ corresponds TF and $w_{\mathbf{Q}.\psi,t}$ corresponds to IDF.

B. ALGORITHMS

B.1 The Two Baseline Algorithms

Baseline 1: This algorithm computes PR scores of all objects and uses an R*-tree to incrementally compute the nearest neighbor in a second stage.

When computing the PR scores, the algorithm obtains a list L_{PR} that ranks the objects involved in ascending order of their scores. The algorithm then incrementally finds nearest neighbors [14] using the R*-tree and checks the PR scores of the objects in L_{PR}, until further objects will not become top-k results.

The tricky part is when to stop finding nearest neighbors. The algorithm maintains the minimum PR score in L_{PR}, denoted by min_{PR}, that has not been "seen" so far, and it maintains the combined ranking score (defined in Equation 3) of the current k-th object, denoted by ξ.

For a newly "seen" object with spatial distance d, if the combined score (the lower the score, the better) computed from d and the current min_{PR} exceeds ξ, the algorithm stops since it is guaranteed that all "unseen" objects will not have lower scores than the current k-th object (and thus cannot be in the result).

Baseline 2. This algorithm computes the PR scores of all objects, thus obtaining a list L_{PR} that ranks them in ascending order of their scores. The list is then scanned to compute the spatial proximity to the query until further scanning will not generate top-k results.

During the scan, the algorithm keeps track of the combined ranking score of the current k-th object, denoted by ξ. For a new object o, if its PR score exceeds ξ, the algorithm stops since all objects after o in L_{PR} will have a score that exceeds ξ; otherwise, we retrieve its location, compute its combined ranking score (Equation 3), and compare with ξ to update ξ if needed.

B.2 Pseudo-Code of ES-EBC

The pseudo-code of Early Stop EBC (ES-EBC) is shown in Algorithm 2. The variables $\vec{p}$, $\vec{q}$, and $outPR$ are as in Algorithm 1. We use a priority queue L_r with the ranking score as its key to keep track of the current top-$(k+1)$ objects, and we use a vector $\vec{L_d}$ to store the distances of the objects to the query.

We use an R*-tree index to incrementally find the next nearest object b to query Q (line 7). Termination occurs when the smallest possible ranking score (its distance to query) of the next object is larger than the current ranking score of the k-th object (lines 8–10).

We check whether the graph block containing b can be pruned according to Theorem 2 (lines 11–13). If the block graph bg_i cannot be pruned, we do the propagation within the block using the same propagation mechanism as in lines 8–23 of Algorithm 1, and maintain the top-$(k+1)$ objects.

If object o is a nearest neighbor that has been accessed (i.e., $\vec{L_d}(o) > 0$), we compute its ranking score and update L_r (lines 16–18).

After we find the set of candidate top-k objects, the algorithm proceeds to propagate the outgoing PR of each block in the graph (lines 19–34). When the upper bound of the k-th object is smaller than the lower bound of the $(k+1)$-st object (line 32), we will stop the propagation and return the top-k objects in L_r according to Theorem 1. Note that although Theorem 2 is used in line 13, it is also applied in line 25 to prune block graphs since $outPR(gb_i)$ can be changed with the propagation.

Algorithm 2 ES-EBC $(rtreeIndex, k, Q)$

```
Input: query $Q$, R*-tree $rtreeIndex$, result size $k$
Output: top-$k$ objects $L_r$
 1: compute the text relevance of each object to $Q$ and $\vec{u_Q}$
 2: $\vec{q} \leftarrow \vec{u_Q}, \vec{p} \leftarrow \vec{0}, L_d \leftarrow \vec{0}, outPR \leftarrow \vec{0}$
 3: for each object $b$ do
 4:     $outPR(b.block) \leftarrow outPR(b.block) + \vec{q}(b)$
 5: $L_r \leftarrow$ NewPriorityQueue(); Initialize $L_r$ with $k+1$ objects whose
    key values are $\infty$
 6: while true do
 7:     $b \leftarrow rtreeIndex$.NextNearestNeighbor($Q$)
 8:     $L_d(b) \leftarrow \mathrm{Dist}(Q, b)$
 9:     if $\mathrm{CRS}(L_r(k)) \le \beta * \frac{L_d(b)}{maxD}$ then
10:         break
11:     $bg_i \leftarrow b.block$
12:     $O_s \leftarrow$ furthest object in the current top-$k$
13:     $pruneDist \leftarrow \lambda \log_{1-\alpha} \frac{\epsilon}{outPR(bg_i)} + L_d(O_s)$
14:     if $bg_i$ is not processed AND
          $\mathrm{Dist}(Q, bg_i) < pruneDist$ then
15:         do propagation on $bg_i$ as in lines 8–23 of Algorithm 1
16:         for each object $o$ s.t. $o \in bg_i$ and $L_d(o) > 0$ do
17:             $\mathrm{CRS}(o) \leftarrow (1-\beta) * (1 - \vec{p}(o)) + \beta * \frac{L_d(o)}{maxD}$
18:             update $L_r$ with $o$ and $\mathrm{CRS}(o)$
19: $blockQueue \leftarrow$ NewPriorityQueue()
20: for each block $bg$ do
21:     $blockQueue$.Enqueue($bg$)
22: while $\|\vec{q}\|_1 \ge \varepsilon$ do
23:     $bg_i$ = $blockQueue$.Dequeue()
24:     $O_s \leftarrow$ the furthest object in the current top-$k$
25:     $pruneDist \leftarrow \lambda \log_{1-\alpha} \frac{\epsilon}{outPR(bg_i)} + L_d(O_s)$
26:     $outPR(bg_i) \leftarrow 0$
27:     if $\mathrm{Dist}(Q, bg_i) < pruneDist$ then
28:         do propagation on $bg_i$ as in lines 8–23 of Algorithm 1
29:         for each object $o$ s.t. $o \in bg_i$ and $L_d(o) > 0$ do
30:             $\mathrm{CRS}(o) \leftarrow (1-\beta) * (1 - \vec{p}(o)) + \beta * \frac{L_d(o)}{maxD}$
31:             update $L_r$ with $o$ and $\mathrm{CRS}(o)$
32:     if $\mathrm{CRS}(L_r(k)) < \mathrm{CRS}(L_r(k+1)) - (1-\beta) * (\alpha^2 max(\vec{q}) +$
        $(1-\alpha)\|\vec{q}\|_1)$ then
33:         break;
34:     $blockQueue$.Update()
```

B.3 Pseudo-Code of S-EBC

The pseudo-code of Subgraph-based EBC (S-EBC) is given in Algorithm 3.

We use $\vec{p}$ to store the current PR score of each object, $\vec{s}$ to store the accumulated PR score (to be distributed) of the border nodes, and a priority queue $queue$ to keep track of the nodes to be visited (lines 1–3).

In each step, we dequeue a node X from $queue$. If the minimum possible ranking score $spRS$ of this node exceeds the ranking score of the current k-th object, we terminate the algorithm and return the results (lines 7–8).

If the node is a non-leaf node, we compute the smallest possible ranking score for each of its child nodes and enqueue them in $queue$ (lines 10–12). Otherwise, we process the subgraph corresponding to the leaf node X by doing the PR score propagation within the subgraph and accumulating the PR scores for border objects (lines 17–27).

In lines 28–32 the global PR score vectors of border objects are used to distribute the accumulated PR scores at the border objects. We update the rankings of all the objects that have been accessed (lines 33–35) because the propagation in the current subgraph $\mathcal{G}_i$ may increase the PR of the objects in other subgraphs. To ensure that the maximal possible error in the ranking score of each top-k object is smaller than the error bound σ, we do postprocessing as discussed in Section 3.3.4 (line 36).

Algorithm 3 S-EBC $(index, k, Q)$

```
Input: query $Q$, IR-tree $index$, result size $k$,
Output: top-$k$ objects $L_r$
1:  $\vec{p} \leftarrow \vec{0}$, $\vec{s} \leftarrow \vec{0}$, $seenObjects = \emptyset$, and initialize a list $L_r$
2:  $queue \leftarrow$ NewPriorityQueue()
3:  $queue$.Enqueue($index.rootNode, 0$)
4:  while not $queue$.Empty() do
5:      $X \leftarrow queue$.Dequeue()
6:      $spRS \leftarrow \text{minRS}(Q, X)$
7:      if $spRS \geq L_r(k)$ then
8:          break
9:      if $X$ is a non-leaf node then
10:         for each entry $child$ in $X$ do
11:             $spRS \leftarrow \text{minRS}(Q, child)$
12:             $queue$.Enqueue($child, spRS$)
13:     else
14:         read the corresponding graph $\mathcal{G}_i$ of $X$
15:         $seenObjects \leftarrow seenObjects \cup \{o | o \in X\}$
16:         $\vec{q} \leftarrow \vec{u}(\mathcal{G}_i)$
17:         while $\|\vec{q}\|_1 \geq \varepsilon$ do
18:             Pick an object $b$ in $\mathcal{G}_i$
19:             if $b \in border(\mathcal{G}_i)$ then
20:                 $\vec{s}(b) \leftarrow \vec{s}(b) + \vec{q}(b)$
21:             else
22:                 $\vec{p}(b) \leftarrow \vec{p}(b) + \alpha\vec{q}(b)$
23:                 for each out-neighbor $j$ of $b$ do
24:                     $\vec{q}(j) \leftarrow \vec{q}(j) + (1-\alpha)C(b,j)\vec{q}(b)$
25:                 $\text{CRS}(b) \leftarrow (1-\beta)(1-\vec{p}(b)) + \beta\frac{\text{Dist}(Q,b)}{maxD}$
26:                 update the position of $b$ in $L_r$
27:             $\vec{q}(b) \leftarrow 0$
28:         for each object $o$ in $border(\mathcal{G}_i)$ do
29:             read the global PR vector $\vec{GP}_o$ of object $o$
30:             $\vec{p} \leftarrow \vec{p} + \vec{s}(o)\vec{GP}_o$
31:             $\text{CRS}(o) \leftarrow (1-\beta)(1-\vec{p}(o)) + \beta\frac{\text{Dist}(Q,o)}{maxD}$
32:             update the position of $o$ in $L_r$
33:         for each node $o$ in $seenObjects$ do
34:             $\text{CRS}(o) \leftarrow (1-\beta)(1-\vec{p}(n)) + \beta\frac{\text{Dist}(Q,o)}{maxD}$
35:             update the position of $o$ in $L_r$
36: do the postprocessing
37: return $L_r$
```

C. PROOFS OF LEMMAS, THEOREMS

Proofs of Lemmas 1 and 2

Lemma 1 holds because in each PR scoring iteration, a node will increase its current PR score. The proof of Lemma 2 follows from related work [13].

Proof of Theorem 1

Given a query Q, we have $\text{RS}(Q, L_r(k+1)) \geq \text{Lower}(L_r(k+1))$ and $\text{RS}(Q, L_r(k)) \leq \text{Upper}(L_r(k))$.

Together with the condition in the theorem, we also have that $\text{RS}(Q, L_r(k)) < \text{RS}(Q, L_r(k+1))$. Consider an object m in $L_r[1, k-1]$ and an object n not in L_r. Because $\text{CRS}(m) \leq \text{CRS}(L_r(k)) \leq \text{CRS}(L_r(k+1)) \leq \text{CRS}(n)$ and because of Equation 4, we have $\text{Upper}(m) \leq \text{Upper}(L_r(k) < \text{Lower}(L_r(k+1)) \leq \text{Lower}(n)$. Thus, we have $\text{RS}(Q, m) < \text{RS}(Q, n)$.

Proof of Theorem 2

According to the triangle inequality, we have $\text{Dist}(o_s, \Omega_i) \geq |\text{Dist}(Q, \Omega_i) - \text{Dist}(Q, o_s)|$. Hence, the minimum number of edges in the path from O_s to gb is $\frac{|\text{Dist}(Q,\Omega_i)-\text{Dist}(Q,o_s)|}{\lambda}$.

In each propagation, an object will distribute the fraction $(1-\alpha)$ of it PR score to other objects by following its out-edges. Therefore, it follows that an upper bound on the effect of Ω_i on o_s is $outPR(\Omega_i)(1-\alpha)^{\frac{|\text{Dist}(Q,gb)-\text{Dist}(Q,o_s)|}{\lambda}}$. If the upper bound is smaller than ϵ, the effect of graph block Ω_i on o_s can be ignored (the effect on other objects in C is even smaller). From the inequality relationship between the upper bound and ϵ, when a block graph is far away, we can get:

$$\text{Dist}(Q, \Omega_i) > \lambda \log_{1-\alpha} \frac{\epsilon}{outPR(\Omega_i)} + \text{Dist}(Q, o_s),$$

which complete the proof.

Proof of Lemma 3

We have that $\vec{\text{Pr}}_b = \vec{\text{Pr}}_b(\mathcal{G}_i) + \vec{\text{Pr}}_b(\mathcal{G} - \mathcal{G}_i)$, where $\vec{\text{Pr}}_b(\mathcal{G}_i)$ represents the distribution to nodes excluding the border nodes in $\mathcal{G}_i$, and $\vec{\text{Pr}}_b(\mathcal{G} - \mathcal{G}_i)$ represents the distribution to the rest nodes.

When we finish distributing the PR in subgraph $\mathcal{G}_i$, all the border nodes in $\mathcal{G}_i$ hold their accumulated PR scores that have not yet distributed to other subgraphs. The nodes in other subgraphs ($\mathcal{G}$ - $\mathcal{G}_i$) are affected by these border nodes, and thus $\vec{\text{Pr}}_b(\mathcal{G} - \mathcal{G}_i) = \sum_{h \in border(\mathcal{G}_i)} \vec{\text{AP}}_b(h) \cdot \vec{\text{Pr}}_h(\mathcal{G} - \mathcal{G}_i)$. The PR scores of the nodes in $\mathcal{G}_i$ come from two parts: the PR distribution within $\mathcal{G}_i$, and the distribution of the accumulated PR scores of the border nodes in $\mathcal{G}_i$. Hence, we have $\vec{\text{Pr}}_b(\mathcal{G}_i) = \vec{\text{LP}}_b + \sum_{h \in border(\mathcal{G}_i)} \vec{\text{AP}}_b(h) \cdot \vec{\text{Pr}}_h(\mathcal{G}_i)$. We get the proof by adding $\vec{\text{Pr}}_b(\mathcal{G}_i)$ and $\vec{\text{Pr}}_b(\mathcal{G} - \mathcal{G}_i)$.

Proof of Theorem 3

Similar to the linearity property of PPV [16], the linearity property also holds for the PR score vector. Hence, we can compute $\vec{\text{Pr}}_Q$ as follows:

$$\vec{\text{Pr}}_Q = \sum_{O \in \mathcal{D}} \text{Sim}(Q, O)\vec{\text{Pr}}_O$$

According to Lemma 3, we obtain:

$$\begin{aligned}\vec{\text{Pr}}_Q &= \sum_{j=1}^{m} \sum_{O \in \mathcal{G}_j} \text{Sim}(Q, O)\vec{\text{Pr}}_O \\ &= \sum_{j=1}^{m} \sum_{O \in \mathcal{G}_j} \text{Sim}(Q, O)(\vec{\text{LP}}_O + \sum_{h \in border(\mathcal{G}_j)} \vec{\text{AP}}_O(h) \cdot \vec{\text{Pr}}_h)\end{aligned}$$

Proof of Lemma 4

The largest PR score is generated in the following situation:

One node has the largest initial PR $\max_{O \in \mathcal{G}_i}(\text{Sim}(t, O))$. It distributes this value, and its out-neighbors gain at most $\text{IPS}(t, \mathcal{G}_i) - \alpha \max_{O \in \mathcal{G}_i}(Sim(t, O))$ PR; then the out-neighbors propagate at most $(1-\alpha)$ $(\text{IPS}(t, \mathcal{G}_i) - \alpha \max_{O \in \mathcal{G}_i}(\text{Sim}(t, O)))$ to the node, and the node keeps the fraction α of this value, and it distributes the fraction $(1-\alpha)$. In the next propagation, the out-neighbors send back $(1-\alpha)^3(\text{IPS}(t, \mathcal{G}_i) - \alpha \max_{O \in \mathcal{G}_i}(\text{Sim}(t, O)))$. This process continues until no PR needs to be propagated. The total PR the node finally holds is:

$$\begin{aligned}\text{maxLPr}(t,\mathcal{G}_i) &= \alpha \max_{O\in\mathcal{G}_i}(\text{Sim}(t,O)) + \alpha(((1-\alpha)+...+\\ (1-\alpha)^{2n+1}&+...)(\text{IPS}(t,\mathcal{G}_i) - \alpha\max_{O\in\mathcal{G}_i}(\text{Sim}(t,O))))\\ &=\alpha\max_{O\in\mathcal{G}_i}(\text{Sim}(t,O))\\ &+\frac{\alpha*(1-\alpha)}{1-(1-\alpha)^2}(\text{IPS}(t,\mathcal{G}_i)-\alpha\max_{O\in\mathcal{G}_i}(\text{Sim}(t,O))))\\ &=\frac{1+\alpha-\alpha^2}{2-\alpha}\max_{O\in\mathcal{G}_i}(\text{Sim}(t,O))+\frac{1-\alpha}{2-\alpha}\text{IPS}(t,\mathcal{G}_i)\end{aligned}$$

Proof of Theorem 4

According to Lemma 3, the largest PR score from the propagation within $\mathcal{G}_i$ is $\text{maxLPr}(t,\mathcal{G}_i)$.

We next consider the effect of other subgraphs. The maximum effect of subgraph $\mathcal{G}_j$ on $\mathcal{G}_i$ occurs if a certain border node in $\mathcal{G}_j$ gets the initial prestige $\text{IPS}(t,\mathcal{G}_i)$. This is because all the effect of $\mathcal{G}_j$ on $\mathcal{G}_i$ is from border nodes. Each global PR vector of a border node in $\mathcal{G}_j$ describes its effect on nodes in $\mathcal{G}_i$. We find the largest value from all the global PR vectors of border nodes, i.e., $\max_{bn\in border(\mathcal{G}_j),b\in\mathcal{G}_i}(\vec{\text{Pr}}_{bn}(b))$, and this is the maximum possible PR that can be propagated from $\mathcal{G}_j$ to a node in $\mathcal{G}_i$.

Combining the two parts completes the proof.

Proof of Lemma 5

$$\begin{aligned}\text{IPS}(Q,\mathcal{G}_i) &= \sum_{O\in\mathcal{G}_i}\text{Sim}(Q,O)\\ &=\sum_{O\in\mathcal{G}_i}\sum_{t\in Q.\psi\bigcap O.\psi}\frac{w_{\mathbf{Q}.\psi,t}w_{\mathbf{O}.\psi,t}}{W_{\mathbf{Q}.\psi}W_{\mathbf{O}.\psi}}\\ &=\sum_{t\in Q.\psi\bigcap\mathcal{G}_i.\psi}\sum_{O\in\mathcal{G}_i}\frac{w_{\mathbf{Q}.\psi,t}w_{\mathbf{O}.\psi,t}}{W_{\mathbf{Q}.\psi}W_{\mathbf{O}.\psi}}\\ &=\sum_{t\in Q.\psi\bigcap\mathcal{G}_i.\psi}\frac{w_{\mathbf{Q}.\psi,t}}{W_{\mathbf{Q}.\psi}}\sum_{O\in\mathcal{G}_i}\frac{w_{\mathbf{O}.\psi,t}}{W_{\mathbf{O}.\psi}}\end{aligned}$$

and we know $\text{Sim}(t,O)=\frac{w_{t,t}w_{\mathbf{O}.\psi,t}}{w_{t,t}W_{\mathbf{O}.\psi}}=\frac{w_{\mathbf{O}.\psi,t}}{W_{\mathbf{O}.\psi}}$, so:

$$\begin{aligned}\text{IPS}(Q,\mathcal{G}_i) &= \sum_{t\in Q.\psi\bigcap\mathcal{G}_i.\psi}\frac{w_{\mathbf{Q}.\psi,t}}{W_{\mathbf{Q}.\psi}}\sum_{O\in\mathcal{G}_i}\text{Sim}(t,O)\\ &=\sum_{t\in Q.\psi\bigcap\mathcal{G}_i.\psi}\frac{w_{\mathbf{Q}.\psi,t}}{W_{\mathbf{Q}.\psi}}\text{IPS}(t,\mathcal{G}_i)\end{aligned}$$

Proof of Theorem 5

Given any object $o\in XO$ and any term t, it holds true that $\text{maxGPr}(t,X)\geq\text{Pr}(t,o)$. Therefore,

$$\begin{aligned}\text{maxPr}(Q,X) &= \sum_{t\in Q.\psi\bigcap X.\psi}\frac{w_{Q.\psi,t}}{W_{\mathbf{Q}.\psi}}\text{maxGPr}(t,X)\\ \geq&\sum_{t\in Q.\psi\bigcap o.\psi}\frac{w_{Q.\psi,t}}{W_{\mathbf{Q}.\psi}}\text{Pr}(t,o)=\text{Pr}(Q,o)\end{aligned}$$

Proof of Theorem 6

Given any object $o\in XO$, we know that $\text{maxPr}(Q,X)\geq\text{Pr}(Q,o)$, according to Theorem 5. Because o is contained in the region $X.\Omega$, we have $\text{Dist}(Q.\mu,X.\Omega)\leq\text{Dist}(Q,o)$. Hence we get $\text{minRS}(Q,X)\leq\text{RS}(Q,o)$.

Proof of Theorem 7

It holds that the maximum possible effect of $\mathcal{G}_j$ on $\mathcal{G}_i$ is a factor of $\max_{bn\in border(\mathcal{G}_j),b\in\mathcal{G}_i}(\vec{\text{Pr}}_{bn}(b))$. Multiplied by the total prestige of $\mathcal{G}_j$, we can get the maximum PR that $\mathcal{G}_j$ can propagate to an object in $\mathcal{G}_i$.

D. SUPPLEMENTARY EXPERIMENTS

D.1 Additional Dataset Details

Dataset **GN** is from the U.S. Board on Geographic Names (geonames.usgs.gov). An object is a location with a geographic name. Dataset **Web** is generated from two datasets. One is WEBSPAM-UK2007[1] that consists of a large number of web documents; the other is a spatial dataset containing the tiger Census blocks in Iowa, Kansas, Missouri, and Nebraska (www.rtreeportal.org). We randomly combine web documents and spatial objects to get the Web dataset. Dataset **Hotel** contains spatial objects that represent hotels in the US (www.allstays.com). Each object has a location and a set of words that describe the hotel (e.g., restaurant, pool).

D.2 Experiments on Web and Hotel

Due to space limitations, we only give the results when varying k and the number of keywords. Figures 7-10 show that ES-EBC and S-EBC significantly outperform the baseline on the two datasets.

D.3 Space Requirements

Table 2 shows the total sizes of the index structures used by each method for data set GN. Baseline and ES-EBC use the same indexes (inverted list and R*-tree) and object graph. S-EBC needs more disk space to store the PR vectors for border objects and the inverted lists in non-leaf nodes. The inverted files in the leaf nodes of the IR-tree are roughly the inverted file used in the baseline approach.

Baseline	ES-EBC	S-EBC
201	201	1423

Table 2: Index structure sizes (MB)

D.4 Effectiveness

To study the utility of L*k*PT queries, we compare with the L*k*T query [8]. The difference between L*k*PT and L*k*T [8] is that the former considers the effect of nearby relevant objects, while the latter does not.

The utility of L*k*PT queries. The lack of a publicly available test data, including both annotated resources and relevant queries, renders the comparison of the different approaches particularly challenging. To enable comparison, we collected a real spatial data set from the region of Aalborg, Denmark using a local Yellow Page service (www.degulesider.dk), where each object has category (e.g., restaurant, hotel) and a description; we geocoded the objects using the Google Maps API. The dataset contains 4,951 objects with a total of 39,505 descriptive words. This dataset has the benefit that we can find expert annotators for it.

We randomly generate 50 locations in the space and ask annotators to choose keywords for each, thus obtaining 50 queries.

To evaluate the quality of query results, we use a well-known metric, the nDCG [15]. The top 5 objects returned by ES-EBC, S-EBC, and L*k*T [8] are merged into a single list, shuffled, and then given to three annotators for judgment. Numerical scores of 0, 1, 2, and 3 are collected and averaged to reflect the annotators' opinions as to whether an object belongs in the top 5.

In L*k*PT (ES-EBC and S-EBC), α and β are set as to their default value of 0.5, and in L*k*T [8], the parameter that balances distance and text relevance (corresponding to β in L*k*PT) is set to 0.5.

[1]barcelona.research.yahoo.net/webspam/datasets/uk2007

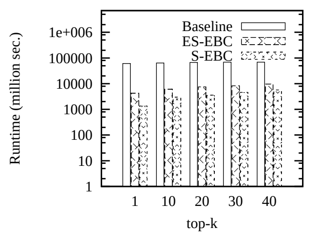

Figure 7: Varying k (Web)

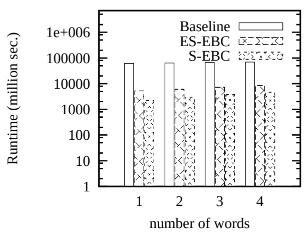

Figure 8: Varying # keywords (Web)

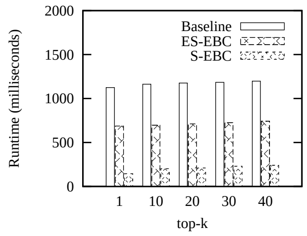

Figure 9: Varying k (Hotel)

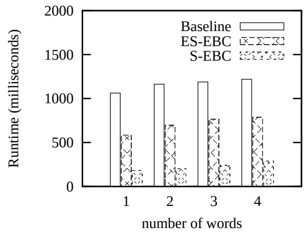

Figure 10: Varying # keywords (Hotel)

Table 3 depicts the results. Both ES-EBC and S-EBC perform significantly better than LkT queries that do not take into account the effects of nearby relevant objects. The approximate S-EBC algorithm performs slightly worse than ES-EBC.

	ES-EBC	S-EBC	LkT [8]
nDCG@5	0.8873	0.8524	0.7061

Table 3: Effectiveness of different algorithms

D.5 Effects of Parameters on Graph Building

Figures 11 and 12 show the runtime when we vary λ and ξ on Hotel. The runtime increases as we increase λ or decrease ξ. The reason is that the graphs become denser, making it take longer to propagate PR scores.

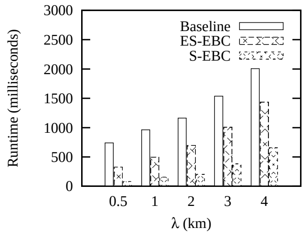

Figure 11: Varying threshold λ (Hotel)

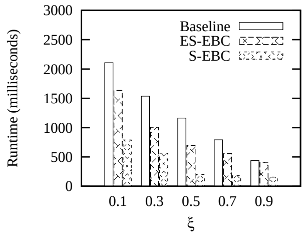

Figure 12: Varying threshold ξ (Hotel)

E. ADDITIONAL RELATED WORK

Spatial Keyword Search: Zhou et al. [22] and Chen et al. [7] handle the problem of retrieving web documents relevant to a keyword query within a pre-specified spatial region. The query processing there occurs in two stages: One type of indexing (e.g., inverted list) is used to filter web document in the first stage, and then another index (e.g., R-tree) is employed in the second stage.

Felipe et al. [10] propose a hybrid index structure that smartly integrates the R-tree with signature files. The hybrid index structure enables to utilize both spatial information and text information to prune the search space at query time. However, this proposal is limited by its use of signature files (e.g., the number of false matches is linear in the collection size and there is no sensible way of using signature files for handling ranking queries [23]).

A hybrid index structure that combines the R*-tree and bitmap indexing is developed to process a new type of query called the m-closest keyword query [21] that returns the closest objects containing at least m keywords. This index structure exhibits the same problems as does the signature-file based indexing [10].

The hybrid index structure called the IR-tree [8] integrates the R-tree and inverted files to enable the efficient processing of the location-aware top-k ranking query by utilizing both location and text information to prune the search space.

Personalized PageRank: In contrast to PageRank [5] that computes the global importance of nodes in a graph, personalized PageRank [16] allows users to favor a set P of preferred nodes. The nodes in the preference set make a unit preference vector u where $u(p) = 1/|P|$ if $p \in P$ and $u(p) = 0$ if $p \notin P$, rather than distributing the unit preference score uniformly over all nodes in PageRank.

Several algorithms [4,12,16] have been proposed to compute the personalized PageRank vector (PPV). Jeh and Widom propose a remarkable Hub Decomposition algorithm [16] that pre-computes the partial vectors for the nodes in a hub set of top-ranked pages. This algorithm can only compute the PPVs of the nodes in the hub set. To process the LkPT query, we need to pre-compute PR for every node, which renders the Hub Decomposition algorithm impractical in our problem.

Fogaras et al. [12] propose a fingerprint-based algorithm that simulates random walks. The idea is to compute and store short random walks from each node in order to compute PPVs at query time. This works well to compute random walks from every node in the graph. However, this cannot be applied to computing a random walk from an arbitrary node, which is prohibitive at query time. This renders the proposal impractical for computing the PR scores in our problem.

More recently, Berkhin proposes a bookmark-coloring algorithm (BCA) [4] that perhaps fits the best with our problem among the existing algorithms for computing PPVs. Its main idea is to diffuse scores in preference vector across the graph. A unit amount of score (called paint) is injected into a selected node (the bookmark node); a fraction of the paint is held by this node, and the rest flows by following the links of the graph. This propagation continues until the paint is distributed over the whole graph. In practice, the algorithm terminates when the paint to be distributed is smaller than a threshold.

PPVs are also ised for keyword queries in entity-relation graphs. In ObjectRank [1], a PPV for each keyword in a graph database is pre-computed. However, it is impractical to pre-compute the PPVs for each keyword when the vocabulary size is large [6].

Chakrabarti [6] apply and extend PPVs to the keyword query on entity-relation graphs. This work is novel in how it chooses a set of nodes as hub nodes based on query logs; and it adopts the approach of Fogaras et al. [12] to store approximate PPVs in the form of fingerprints.

Block PageRank: There is a large body of work on global PageRank computation. Some works consider computations of global PageRank values over subgraphs (e.g., [2, 17]). The problem of computing global PageRank is different from computing Prestige-base Relevance, and these proposals are therefore not directly applicable to our problem.

A final note is that two works [9, 18] that employ the PageRank algorithm to do propagation on document similarity graphs focus on effectiveness without considering efficiency issues.