Location Aware Keyword Query Suggestion Based on Document Proximity

Shuyao Qi, Dingming Wu, and Nikos Mamoulis

Abstract—Keyword suggestion in web search helps users to access relevant information without having to know how to precisely express their queries. Existing keyword suggestion techniques do not consider the locations of the users and the query results; i.e., the spatial proximity of a user to the retrieved results is not taken as a factor in the recommendation. However, the relevance of search results in many applications (e.g., location-based services) is known to be correlated with their spatial proximity to the query issuer. In this paper, we design a location-aware keyword query suggestion framework. We propose a weighted keyword-document graph, which captures both the semantic relevance between keyword queries and the spatial distance between the resulting documents and the user location. The graph is browsed in a random-walk-with-restart fashion, to select the keyword queries with the highest scores as suggestions. To make our framework scalable, we propose a partition-based approach that outperforms the baseline algorithm by up to an order of magnitude. The appropriateness of our framework and the performance of the algorithms are evaluated using real data.

1 Introduction

Users often have difficulties in expressing their web search needs; they may not know the keywords that can retrieve the information they require [1]. *Keyword suggestion* (also known as query suggestion), which has become one of the most fundamental features of commercial Web search engines, helps in this direction. After submitting a keyword query, the user may not be satisfied with the results, so the keyword suggestion module of the search engine recommends a set of m keyword queries that are most likely to refine the user's search in the right direction. Effective keyword suggestion methods are based on click information from query logs [2], [3], [4], [5], [6], [7], [8] and query session data [9], [10], [11], or query topic models [12]. New keyword suggestions can be determined according to their semantic relevance to the original keyword query. The semantic relevance between two keyword queries can be determined (i) based on the overlap of their clicked URLs in a query log [2], [3], [4], (ii) by their proximity in a bipartite graph that connects keyword queries and their clicked URLs in the query log [5], [6], [7], [8], (iii) according to their co-occurrences in query sessions [13], and (iv) based on their similarity in the topic distribution space [12].

However, none of the existing methods provide *location-aware* keyword query suggestion, such that the suggested keyword queries can retrieve documents not only related to the user information needs but also located near the user location. This requirement emerges due to the popularity of spatial keyword search [14], [15], [16], [17], [18] that takes a user location and user-supplied keyword query as arguments and returns objects that are spatially close and textually relevant to these arguments. Google processed a daily average of 4.7 billion queries in 2011[1], a substantial fraction of which have local intent and target spatial web objects (i.e., points of interest with a web presence having locations as well as text descriptions) or geo-documents (i.e., documents associated with geo-locations). Furthermore, 53% of Bing's mobile searches in 2011 were found to have a local intent.[2]

To fill this gap, we propose a Location-aware Keyword query Suggestion (LKS) framework. We illustrate the benefit of LKS using a toy example. Consider five geo-documents d_1–d_5 as listed in Figure 1(a). Each document d_i is associated with a location $d_i.\lambda$ as shown in Figure 1(b). Assume that a user issues a keyword query k_q = "*seafood*" at location λ_q, shown in Figure 1(b). Note that the relevant documents d_1–d_3 (containing "*seafood*") are far from λ_q. A location-aware suggestion is "*lobster*", which can retrieve nearby documents d_4 and d_5 that are also relevant to the user's original search intention. Previous keyword query suggestion models (e.g., [6]) ignore the user location and would suggest "*fish*", which again fails to retrieve nearby relevant documents. Note that LKS has a different goal and therefore differs from other location-aware recommendation methods (e.g., auto-completion/instant search [19], [20], tag recommendation [21]). Section 5 provides a detailed discussion about the differences between LKS and these models, while in Section 4 we experimentally show that an adaptation of the method in [21] is less effective than LKS.

The first challenge of our LKS framework is how to effectively measure keyword query similarity while capturing the spatial distance factor. In accordance to previous query suggestion approaches [3], [4], [5], [6], [7], [8], [10], [11], LKS constructs and uses a keyword-document bipartite graph (KD-graph for short), which connects the keyword queries with their relevant documents as shown in Figure 1(c). Different to all previous approaches which ignore locations, LKS *adjusts* the weights on edges in the KD-graph to capture not only the semantic relevance between keyword queries, but also the spatial distance between the document locations and the query issuer's location λ_q. We apply a *random walk with restart* (RWR) process [22] on the KD-graph, starting from the user supplied query k_q, to find the set of m key-

• *S. Qi, D. Wu and N. Mamoulis are with the Department of Computer Science, the University of Hong Kong, Hong Kong*

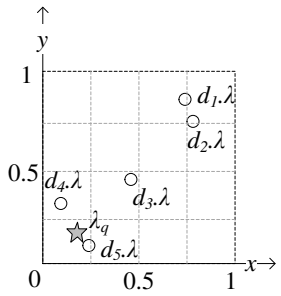
1. http://www.statisticbrain.com/google-searches

2. http://searchengineland.com/microsoft-53-percent-of-mobile-searches-have-local-intent-55556

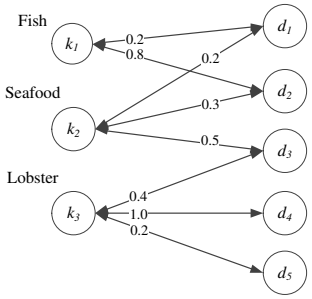
d_1	Fish and Seafood
d_2	Fish Seafood
d_3	Lobster Seafood
d_4	Lobster Restaurant
d_5	Lobster House

k_1	Fish
k_2	Seafood
k_3	Lobster

(a) Documents and Keyword Queries



(b) Locations of Documents



(c) KD-Graph

	λ_q
d_1	1.0
d_2	0.9
d_3	0.5
d_4	0.1
d_5	0.0

(d) Euclidean Distances

Fig. 1. LKS Example

word queries with the highest semantic relevance to k_q and spatial proximity to the user location. RWR on a KD-graph has been considered superior to alternative approaches [7] and has been a standard technique employed in various (location-independent) keyword suggestion studies [5], [6], [7], [8], [10], [11].

The second challenge is to compute the suggestions efficiently on a large dynamic graph. Performing keyword suggestion instantly is important for the applicability of LKS in practice. However, RWR search has a high computational cost on large graphs. Previous work on scaling up RWR search require pre-computation and/or graph segmentation [22], [23], [24], [25], [26]; part of the required RWR scores are materialized under the assumption that the transition probabilities between nodes (i.e., the edge weights) are known beforehand. In addition, RWR search algorithms that do not rely on pre-computation (e.g., [27]) accelerate the computation by pruning nodes based on their lower or upper bound scores and also require the full transition probabilities. However, the edge weights of our KD-graph are unknown in advance, hindering the application of all these approaches. To the best of our knowledge, no existing technique can accelerate RWR when edge weights are unknown apriori (or they are dynamic). To address this issue, we present a novel partition-based algorithm (PA) that greatly reduces the cost of RWR search on such a dynamic bipartite graph. In a nutshell, our proposal divides the keyword queries and the documents into partitions and adopts a lazy mechanism that accelerates RWR search. PA and the lazy mechanism are generic techniques for RWR search, orthogonal to LKS, therefore they can be applied to speed up RWR search in other large graphs.

In summary, the contributions of this paper are:

- We design a Location-aware Keyword query Suggestion (LKS) framework, which provides suggestions that are relevant to the user's information needs and can retrieve relevant documents close to the query issuer's location.
- We extend the state-of-the-art Bookmark Coloring Algorithm (BCA) [28] for RWR search to compute the location-aware suggestions. In addition, we propose a partition-based algorithm PA that computes the scores of the keyword queries at the partition level and adopts a lazy mechanism to greatly reduce the computational cost of BCA.
- We evaluate the effect of locations in LKS through an empirical study and demonstrate that useful location-aware suggestions are provided. We also show experimentally that PA is two times to one order of magnitude faster than BCA.

The rest of the paper is organized as follows. The LKS framework is introduced in Section 2. The baseline algorithm and the partition-based algorithm are presented in Section 3. We evaluate the location effect in the framework and the performance of the algorithms in Section 4. Related work is reviewed in Section 5 and we conclude in Section 6.

2 LKS Framework

Consider a user-supplied query q with initial input k_q; k_q can be a single word or a phrase. Assuming that the query issuer is at location λ_q, two intuitive criteria for selecting good suggestions are: (i) the suggested keyword queries (words or phrases) should satisfy the user's information needs based on k_q and (ii) the suggested queries can retrieve relevant documents spatially close to λ_q. The proposed LKS framework captures these two criteria.

2.1 Keyword-Document Graph

Without loss of generality, we consider a set of geo-documents D such that each document $d_i \in D$ has a point location $d_i.\lambda$.[3] Let K be a collection of keyword queries from a query log. We consider a directed weighted bipartite graph $G = (D, K, E)$ between D and K and refer to it as the keyword-document graph (or simply KD-graph). If a document d_i is clicked by a user who issued keyword query k_j in the query log, E contains an edge e from k_j to d_i and an edge e' from d_i to k_j. Initially, the weights of edges e and e' are the same and equal to the number of clicks on document d_i, given keyword query k_j [2]. Therefore, the direct relevance between a keyword query and a clicked document is captured by the edge weight. Furthermore, the semantic relevance between two keyword queries is captured by their proximity in the graph G (e.g., computed as their RWR distance). Any updates in the query log and/or the document database can be easily applied on the KD-graph; for a new query/document, we add a new node to the graph; for new clicks, we only need to

3. If a document relates to multiple locations, we can model it as multiple documents, each referring to a single location. Location-independent documents can also be included in our framework by turning off the location awareness component for them.

update the corresponding edge weights accordingly. As an example, Figure 1(a) shows five documents d_1–d_5 and three keyword queries k_1–k_3. The corresponding KD-graph is shown in Figure 1(c). For the ease of presentation, the edge weights are normalized (i.e., divided by the maximum number of clicks in the log for any query-document pair).

2.2 Location-aware Edge Weight Adjustment

The initial KD-graph is what a classic keyword suggestion approach would use [5], [6], [7], [8], [10], [11], because it captures the semantics and textual relevance between the keyword query and document nodes; i.e., the first criterion of location-aware suggestion. In order to satisfy the second criterion (i.e., location awareness), we propose to *adjust* the edge weights in the KD-graph based on the spatial relationships between the location of the query issuer and the nodes of the KD-graph. Note that this edge adjustment is query-dependent and dynamic. In other words, different adjustment is used for each different query independently.

We now outline the details of the edge weights adjustment. Recall that a user-supplied query q consists of two arguments: an input keyword query k_q (a word or a phrase) and a query location λ_q. Given q, the weight $w(e)$ of the edge e from a keyword query node k_i to a document node d_j is adjusted by the following function:

$$\tilde{w}(e) = \beta \times w(e) + (1 - \beta) \times (1 - dist(\lambda_q, d_j.\lambda)) \quad (1)$$

where $w(e)$ is the initial weight of e in the KD-graph, $\tilde{w}(e)$ is the adjusted edge weight, $dist(\lambda_q, d_j.\lambda)$ is the Euclidean distance between the query issuer's location λ_q and document d_j, and parameter $\beta \in [0, 1]$ is used to balance the importance between the original (i.e., click-based) weight and the distance of d_j to the query location. Euclidean distances are normalized to take values in $[0, 1]$. This keyword-to-document edge weight adjustment increases the weights of the documents that are close to the user's location.

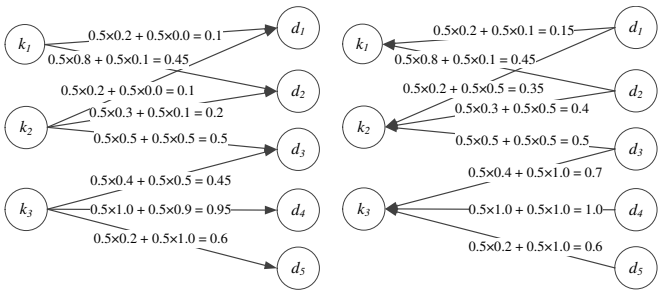
Let $D(k_i)$ be the set of documents connected to a keyword query $k_i \in K$ in the KD-graph. $D(k_i)$ may contain multiple documents and the locations of them form a spatial distribution. We propose to adjust the weights of the edges pointing to k_i by the minimum distance between λ_q and the locations of documents in $D(k_i)$.[4] Such an adjustment favors keyword query nodes which have at least one relevant document close to the query issuer's location λ_q. Specifically, the weight $w(e')$ of the edge e' from a document node d_j to a keyword query node k_i is adjusted as follows:

$$\tilde{w}(e') = \beta \times w(e') + (1 - \beta) \times (1 - mindist(\lambda_q, D(k_i))) \quad (2)$$

where $mindist(\lambda_q, D(k_i))$ is the minimum Euclidean distance[5] between λ_q and any document in $D(k_i)$.

For example, Figure 1(b) shows the locations of the 5 documents of Figure 1(a) and a query location λ_q; Figure 1(d) includes the (approximate) Euclidean distances between λ_q and the five documents. Figure 2 illustrates how the edge weights from keyword query nodes to document nodes (Figure 2(a)) and from document nodes to keyword query nodes (Figure 2(b)) are adjusted based on the query location, assuming $\beta = 0.5$. Take the edge from k_1 to d_1 as a concrete example. Its weight is calculated using Equation 1 where $dist(\lambda_q, d_1.\lambda) = 1$. The weight of the edge from d_1 to k_1 is computed using Equation 2 where $D(k_1) = \{d_1, d_2\}$ and $mindist(\lambda_q, D(k_1)) = 0.9$.

We remark that the original KD-graph G is constructed only once in advance (as in previous work [5], [6], [7], [8], [10], [11]). In addition, any update operations on the KD-graph (discussed in Section 2.1) are independent to our edge weight adjustment strategy, which is query-dependent. Given a user-supplied query q, the adjusted graph G_q is *dynamically* derived from G based on the query location λ_q, used to compute suggestions for q, and then dropped. During this process, G_q is maintained separately and G is not changed, so that concurrent or follow up queries are not affected. As we will discuss in Section 3.1, only a small portion of edges, relevant to the current query, are adjusted and cached, hence the adjustment is conducted efficiently and on-demand, during the keyword query suggestion process.



(a) Keyword queries → Documents (b) Keyword queries ← Documents

Fig. 2. Location-aware Edge Weight Adjustment

2.3 Location-aware Keyword Query Suggestion

We denote by G_q the KD-graph G after adjusting the edge weights, based on the query location λ_q. G_q captures the two criteria of selecting suggestions, i.e., relevance to k_q and closeness to λ_q. Thus, keyword queries close to k_q in G_q are likely to be relevant to k_q and, at the same time, they result in documents close to the query issuer. In order to find the set of keyword queries for recommendation, we compute for all keyword queries a graph proximity score with respect to k_q, based on the random walk with restart (RWR) process. The top-m keyword queries in G_q with the highest scores (excluding k_q) are returned as the suggestions.

Formally, let $\vec{\psi}$ be a column vector recording the RWR scores of all keyword queries in K based on G_q. $\vec{\psi}$ is computed by [29]:

$$\vec{\psi} = (1 - \alpha) M_{DK}^T M_{KD}^T \vec{\psi} + \alpha \vec{\psi}_q \quad (3)$$

M_{DK} is a document-by-keyword matrix and M_{KD} is a keyword-by-document matrix, storing the edge weights in G_q; both matrices are row-normalized. $\vec{\psi}_q$ is the initial score vector having zeros at all positions except the position of k_q, where it has 1. Parameter α represents the probability of a random walker jumping to the initial keyword query

4. Since the locations of past query issuers are not always available (e.g., due to privacy constraints), in this paper, we focus on the case where only document locations are known. Therefore, the edge adjustments for keyword-to-document edges and document-to-keyword edges are performed differently.

5. The effect of using the average distance to $D(k_i)$ is similar.

instead of randomly following an edge in graph G_q. Since the user-supplied query k_q also gets an RWR score, in the end we compute the top-m keyword queries other than k_q.

Example 1. Consider query q with initial input k_q = "*seafood*" and location $\lambda_q = (0.2, 0.2)$ against the documents of Figure 1(a). The locations of the documents and λ_q are shown in Figure 1(b). The KD-graph G is shown in Figure 1(c) and the adjusted edge weights based on λ_q are presented in Figure 2. The top-1 suggestion for q is "*lobster*" according to Equation 3 given $\alpha = 0.5$. Note that "*lobster*" retrieves documents d_4 and d_5 that are close to the query location, which cannot be achieved by "*seafood*". Also queries "*lobster*" and "*seafood*" are semantically relevant.

3 ALGORITHMS

In this section, we introduce a baseline algorithm (BA) to compute the location-aware suggestions (Section 3.1). Then, we propose a more efficient partition-based algorithm (PA) (Section 3.2).

3.1 Baseline Algorithm (BA)

We extend the popular Bookmark-Coloring Algorithm (BCA) [28] to compute the top-m suggestions as a baseline algorithm (BA). BCA models RWR as a *bookmark coloring* process. Starting with one unit of active *ink* injected into node k_q, BA processes the nodes in the graph in descending order of their active ink. Different from typical personalized PageRank problems [27], [30] where the graph is homogeneous, our KD-graph G_q has two types of nodes: keyword query nodes and document nodes. As opposed to BCA, BA only ranks keyword query nodes; a keyword query node *retains* α portion of its active ink and *distributes* $1-\alpha$ portion to its neighbor nodes based on its outgoing adjusted edge weights, while a document node distributes all its active ink to its neighbor nodes.

In our implementation, the weight of each edge e is adjusted based on λ_q *online*, at the time when the source node of e is distributing ink. This means that the edge weight adjustment is done during BA (i.e., G_q needs not be computed and materialized before the algorithm starts). Moreover, a node may be processed several times; thus, the adjusted weights of its outgoing edges are cached after the node is first processed, for later usage. A node can distribute ink when its active ink exceeds a threshold ϵ. Algorithm BA terminates when either (i) the ink retained at the top-mth keyword query node is more than the ink retained at the top-$(m+1)$th keyword query node plus the sum of the active ink of all nodes [30] or (ii) the active ink of each node is less than ϵ (typically, $\epsilon = 10^{-5}$).

Algorithm 1 is a pseudo code of BA. Priority queue Q maintains the nodes to be processed in descending order of their active ink (line 1). Q initially contains one entry, i.e., the user-supplied keywords k_q with active ink 1 (line 2). Priority queue C, initially empty, stores the candidate suggestions in descending order of their retained ink (line 1). The sum of the active ink of all nodes $AINK$ is set to 1 (line 3). Termination conditions (i) and (ii) are checked at lines 4 and 8, respectively. The processing of a keyword query node involves retaining α portion of its active ink (line 13) and distributing $1-\alpha$ portion to each of its neighbor document nodes based on the adjusted edge weights (lines 19–23). The total active ink $AINK$ is modified accordingly (line 14). As soon as a keyword query node has some retained ink, it enters C. The processing of a document node involves distributing all its active ink to neighbor keyword query nodes according to the adjusted edge weights (lines 19–23). The algorithm returns the top-m candidate suggestions other than k_q in C as the result (line 24).

ALGORITHM 1: Baseline BA

```
   Input  : G(D, K, E), q = (k_q, λ_q), m, ε
   Output: C
1  PriorityQueue Q ← ∅, C ← ∅
2  Add k_q to Q with k_q.aink ← 1
3  AINK ← 1
4  while Q ≠ ∅ and Q.top.aink ≥ ε do
5      Deheap the first entry top from Q
6      tm = the top-m entry from C
7      tm' = the top-(m + 1) entry from C
8      if tm.rink > tm'.rink + AINK then
9          break
10     distratio = 1
11     if top is a keyword query node then
12         distratio = 1 − α
13         top.rink ← top.rink + top.aink × α
14         AINK ← AINK − top.aink × α
15         if there exist a copy t of top in C then
16             Remove t from C
17             top.rink ← top.rink + t.rink
18         Add top to C
19     for each node v connected to top in G do
20         v.aink ← top.aink × distratio × w̃(top, v)
21         if there exists a copy v' of v in Q then
22             Remove v' from Q; v.aink ← v.aink + v'.aink
23         Add v to Q
24 return the top-m entries (excluding k_q) in C
```

Example 2. Figure 3 shows the steps of BA (for $m = 1$, $\epsilon = 0.1$ and $\alpha = 0.5$), when applied to the adjusted KD graph of our running example (see Example 1 and Figures 1,2). The number next to each node indicates its amount of active ink. The numbers in rounded rectangles are the amount of retained ink. Initially, one unit amount of ink is injected into node k_2, i.e., the keyword query k_q = "*seafood*" supplied by the user. In the first iteration, node k_2 retains 0.5 amount of ink and distributes 0.5 amount of ink to its neighbor document nodes d_1–d_3 according to the adjusted edge weights. In the second iteration, d_3 distributes its active ink of amount 0.325 to its neighbor keyword query nodes k_2 and k_3. BA terminates at the sixth iteration where the active ink of each node is smaller than ϵ. The top-1 suggestion (excluding user query k_2) is k_3 = "*lobster*", with the largest amount of retained ink (0.098).

3.2 Partition-based Algorithm (PA)

Algorithm BA can be slow for several reasons. First, at each iteration, only one node is processed; thus, the active ink drops slowly and the termination conditions are met

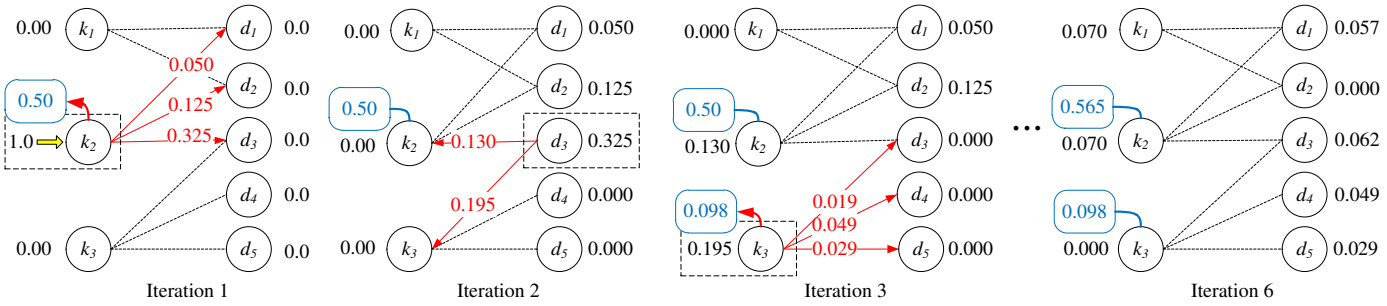


Fig. 3. Illustration of Algorithm BA

after too many iterations. Second, given the large number of iterations, the overhead of maintaining queue Q is significant. Finally, the nodes distribute their active ink to all their neighbors, even if some of them only receive a small amount of ink. We note that existing pre-processing techniques that can accelerate RWR search and BCA (e.g., the pre-selection of *hub* nodes [28]) require complete knowledge of the graph before the algorithm starts. Therefore, they are not applicable to our problem, because the edge weights in graph G_q depend on the query location, which is unknown in advance. Applying a pre-computation technique for all possible query locations (i.e., all possible G_q) has extreme computational and storage requirements.

To improve the performance of BA, in this section, we propose a partition-based algorithm (PA) that divides the keyword queries and the documents in the KD-graph G into groups. Let $\mathcal{P}^K = \{P_i^K\}$ be the partitions of the keyword queries and $\mathcal{P}^D = \{P_i^D\}$ be the document partitions. Algorithm PA follows the basic routine of algorithm BA, but with the following differences:

(1) Node-Partition Graphs. PA uses two directed graphs G^{KP} and G^{DP} constructed offline from the KD-graph G and partitions $\mathcal{P}^K$ and $\mathcal{P}^D$. In graph G^{KP}, a keyword query node k_i connects to a document partition P^D if k_i connects in G to at least one document in P^D. Similarly, in graph G^{DP}, a document node d_j connects to a keyword partition P^K if d_j connects in G to at least one keyword query node k_i. As an example, in Figure 4, the document partitions are $P_1^D = \{d_1, d_2\}$ and $P_2^D = \{d_3, d_4, d_5\}$ and the keyword query partitions are $P_1^K = \{k_1\}$ and $P_2^K = \{k_2, k_3\}$. The edge weights are defined based on graph G_q, computed during the execution of PA. Each edge weight shown in Figure 4 indicates the portion of the ink to be distributed to a partition P from a node v that is the *sum* of the adjusted weights of the edges from node v to the nodes in P according to G_q.

(2) Ink Distribution. In PA, each node distributes its active ink to its neighbor partitions (contrast this to BA, where each node distributes its active ink to each of its neighbor nodes). The priority queue used in BA maintains the nodes that will distribute ink, but the priority queue used in PA records the partitions that will be processed. The ink received by a partition is not spread to the nodes inside the partition until this partition reaches the head of the priority queue. The benefit is that a partition may receive ink from the same node several times while waiting in the queue, so that the nodes in this partition receive ink in batch when this

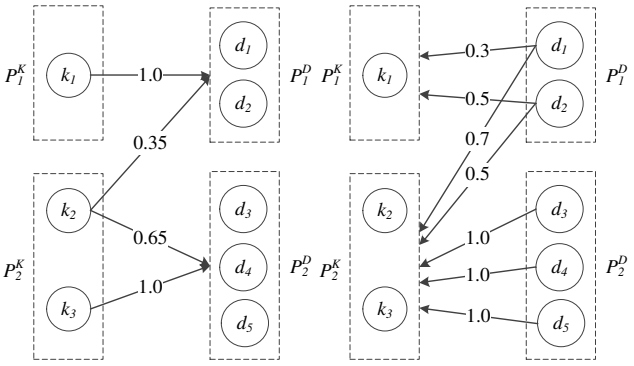
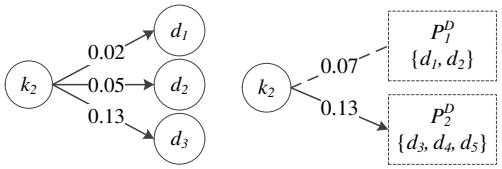


Fig. 4. Node-Partition Graphs

partition reaches the head of the queue. In algorithm PA, the active ink drops fast and the termination conditions may be fulfilled early. Thus, the number of iterations needed is largely reduced and so is the cost spent for maintaining the priority queue Q. Moreover, since the number of partitions is much smaller than that of nodes, the size of queue Q is much smaller compared to that used in BA, so operations on it are fast as well. As an example, in Figure 5, in algorithm BA, node k_2 distributes its active ink to each of its three neighbor nodes d_1–d_3. However, in algorithm PA, the active ink of k_2 is only distributed to two recipients: partitions P_1^D and P_2^D; an underlying document node will not receive the ink, until its partition reaches the top of the queue.

(3) Lazy Distribution Mechanism. In BA, a node distributes ink *aggressively*, i.e., each of its neighbor nodes receives ink no matter how much it is. On the other hand, in algorithm PA, we adopt a lazy distribution mechanism that relies on threshold ϵ. If the amount of the ink to be distributed from a node v to a partition P is smaller than ϵ, P does not receive the ink immediately; instead, the ink is accumulated (i.e., buffered) at v. Later, if at some point the ink accumulated at v for partition P exceeds ϵ, P receives it. Overall, this lazy distribution mechanism delays the distribution of small amounts of ink across the graph that would otherwise result in many updates, reducing the computational cost significantly. As a toy example in Figure 5(b), the amount of ink (0.07) to be distributed from node k_2 to partition P_1^D waits at k_2 when $\epsilon = 0.1$.

Algorithm 2 is a pseudocode for PA. Priority queue Q maintains the partitions to be processed in descending order of their keys (line 1). A partition waiting in the queue may



(a) BA (b) PA

Fig. 5. Ink Distribution Methods

have received ink from multiple nodes. Let $P.aink_{v_i}$ be the ink received by partition P from node v_i. The key of P in the queue is the maximum value of $P.aink_{v_i}$. Priority queue C stores the candidate suggestions in descending order of their retained ink, initialized as empty (line 1). Q initially contains one entry referring to the keyword query partition P containing the user supplied keywords with active ink 1 (line 2). The sum of the active ink of all nodes $AINK$ is set to 1 (line 3). The termination conditions are the same as those of algorithm BA (lines 4 and 8). For a partition P_t being processed, PA firstly spreads its received ink to P_t's nodes (line 10). When processing a keyword query partition, each keyword query node v inside retains α portion of its active ink (line 15). The ranking of this keyword query node in C is updated and the active ink $AINK$ is modified (line 16). For each document partition P_i connected from v in G^{KP}, if the amount of accumulated ink for P_i at v exceeds ϵ, partition P_i receives the ink. Otherwise, the ink is accumulated at v for P_i (lines 24–32). Similarly, when processing a document partition, for each document node v inside the partition, the lazy ink distribution mechanism is applied. As before, the only difference is that document nodes do not retain any ink. The algorithm finally returns the top-m candidate suggestions in C as the result (line 33).

Note that the partitioning is done offline, and thus it does not add any cost to the online query suggestion process. In PA, the cost of maintaining the partition information (in terms of both the memory usage and the CPU cost) is negligible when the number of partitions is small (e.g., 16 – 64 as shown in our experiments).

Example 3. We illustrate algorithm PA (for m=1, $\epsilon = 0.1$, and $\alpha = 0.5$) using our running example ($k_q =$ "*seafood*" and $\lambda_q = (0.2, 0.2)$). Figure 4 shows the node-partition graphs. Figure 6 shows how the scores (ink) of the keyword queries are computed. Initially, partition P_2^K contains the user supplied keyword k_2 having active ink 1. In the first iteration, partition P_2^K is processed. Each keyword query node inside partition P_2^K retains α portion of its active ink. In this example, k_2 retains 0.5 amount of ink. Document partitions P_1^D and P_2^D connected from k_2 according to graph G^{KP} receive 0.175 and 0.325 amount of ink, respectively. In the second iteration, document partition P_2^D is processed. Each node inside P_2^D first receives the ink from k_2 and then distributes ink to the connected keyword query partition P_2^K according to graph G^{DP}. In the third iteration, partition P_2^K is processed. Each node inside P_2^K receives the ink from d_3. After that, node k_3 accumulates ink for partition P_2^D, since the amount of ink for P_2^D is 0.097 ($< \epsilon$). Similarly, node k_2 accumulates ink for partitions P_1^D and P_2^D. In the fourth iteration, partition P_1^D is processed. For each node inside P_1^D, the ink to be distributed does not exceed threshold ϵ, so that d_1 and d_2 accumulate ink for partitions P_1^K and P_2^K. In the end, the top-1 suggestion is keyword query $k_3 =$ "*lobster*".

ALGORITHM 2: PA

Input : $G(D, K, E)$, G^{KP}, G^{DP}, $q = (k_q, \lambda_q)$, m, ϵ
Output: C

1 PriorityQueue $Q \leftarrow \emptyset$, $C \leftarrow \emptyset$
2 Add partition $P \ni k_q$ to Q with $P.aink \leftarrow 1$
3 $AINK \leftarrow 1$
4 **while** $Q \neq \emptyset$ *and* $Q.top.aink_{v_i} \geq \epsilon$ **do**
5 Deheap the top entry P_t from Q
6 tm = the top-m entry from C
7 tm' = the top-$(m+1)$ entry from C
8 **if** $tm.rink > tm'.rink + AINK$ **then**
9 **break**
10 Spread the active ink to nodes in P_t
11 **for** *each node* v *in partition* P_t **do**
12 *distratio* = 1
13 **if** v *is a keyword query node* **then**
14 *distratio* = $1 - \alpha$
15 $v.rink \leftarrow v.rink + v.aink \times \alpha$
16 $AINK \leftarrow AINK - v.aink \times \alpha$
17 **if** *there exist a copy* t *of* v *in* C **then**
18 Remove t from C
19 $v.rink \leftarrow v.rink + t.rink$
20 Add v to C
21 Get partition set $\mathcal{P}$ connected from v in G^{KP}
22 **else**
23 Get partition set $\mathcal{P}$ connected from v in G^{DP}
24 **for** *each partition* P_i *in* $\mathcal{P}$ **do**
25 $ink \leftarrow v.aink \times distratio \times \tilde{w}(v, P_i)$
26 **if** $ink + v.acc.P_i \geq \epsilon$ **then**
27 $P_i.aink \leftarrow ink + v.acc.P_i$
28 **if** *there exist a copy* P_i' *of* P_i *in* Q **then**
29 Remove P_i' from Q;
$P_i.aink \leftarrow P_i.aink + P_i'.aink$
30 Add P_i to Q
31 **else**
32 Accumulate ink at node v for P_i ($v.acc.P_i$)
33 **return** the top-m entries (excluding k_q) in C

Partitioning Methods. We consider four partitioning methods for keyword query/document nodes, to be empirically evaluated in Section 4.

Random Partitioning. Keyword queries are evenly and randomly partitioned into a predetermined number of partitions. The same is done for the documents.

Spatial Partitioning. First, a regular grid is used to partition the Euclidean space of document locations. The documents whose locations lie in the same grid cell form a partition. The keyword queries are partitioned according to the document partitions. The idea is inspired by the *duality of word and document clustering* [31]. Specifically, let $\mathcal{P}^D = \{P_1^D, P_2^D, \cdots, P_n^D\}$ be the document partitions. We initialize N empty keyword query partitions $\mathcal{P}^K = \{P_1^K, P_2^K, \cdots, P_N^K\}$. According to graph G^{KP}, which connects keywords to document partitions, each keyword query node k_i is connected to a set of docu-

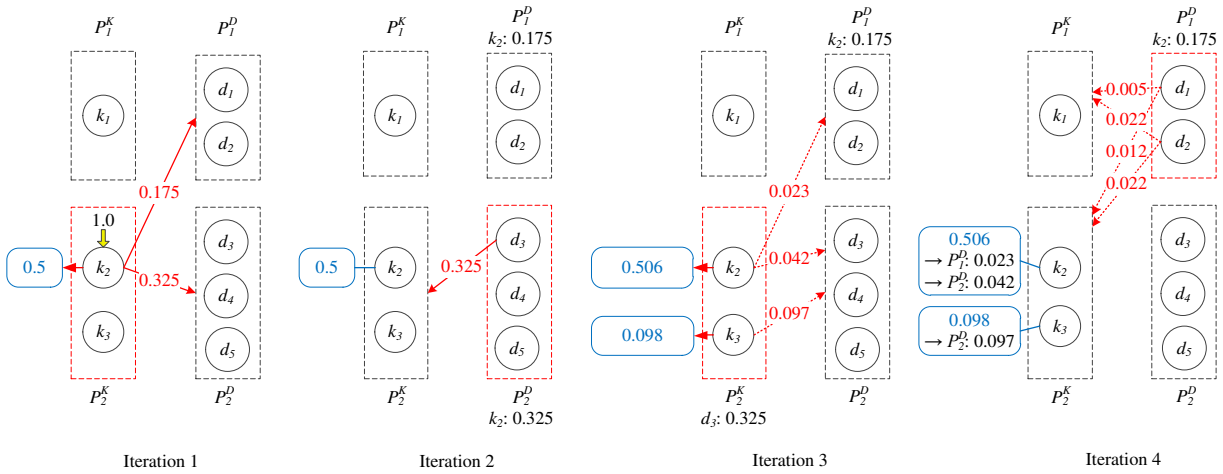


Fig. 6. Illustration of Algorithm PA

ment partitions $P(k_i) = \{P_i^D\}$. Let P_j^D be the document partition connected from k_i with the highest edge weight, i.e., $\arg\max_{P_j^D \in P(k_i)} w(k_i, P_j^D)$. Keyword query node k_i is added to partition P_j^K that has the same subscript j as partition P_j^D. In the end, some of the keyword query partitions in $\mathcal{P}^K$ might be empty, and thus are removed.

Textual Partitioning. Each document d is associated to a vector $d.\psi$ where each dimension refers to a keyword query k connected to d in the KD-graph G and the value of this dimension is the edge weight $w(d, k)$. Let $\cos(d_i.\psi, d_j.\psi)$ be the cosine similarity between the vectors of documents d_i and d_j. A clustering algorithm, e.g., k-means, is applied on the document vectors using the cosine similarity, so that the document partitions are obtained. The keyword queries are partitioned based on the document partitions in the same way as in the spatial partitioning method.

Hybrid Partitioning. Let $f_H(d_i, d_j) = \gamma \times \cos(d_i.\psi, d_j.\psi) + (1-\gamma) \times dist(d_i.\lambda, d_j.\lambda)$ be a hybrid distance between documents d_i and d_j, considering both their Euclidean distance and cosine similarity. A clustering algorithm, e.g., k-means, is applied on the documents using the hybrid distance, so that the document partitions are obtained. The keyword queries are partitioned as in the previous methods.

4 Empirical Study

We conducted a set of experiments in order to evaluate LKS. Section 4.1 presents our experimental setup. The effectiveness of our LKS framework compared to query suggestion that does not consider locations is evaluated in Section 4.2. The runtime performance of algorithms BA and PA is evaluated in Section 4.3.

4.1 Setup

Two datasets AOL and TWEET are used in our evaluation. AOL is based on a real query log of the AOL search engine [32]. Each record in the log contains a keyword query, the time when the query was submitted, and a URL clicked. Since the locations of the URLs are not readily available in the query log, we geo-located the URL domains with the help of the Freegeoip[6] project and removed the URLs without any geo information. In addition, we cleaned the query log by removing the keyword queries without click information and with frequency less than 3. We added an edge between a keyword query node k_i and a URL node d_j if there exist records containing both k_i and d_j in the query log. The edge weight was defined by the number of records containing both k_i and d_j (i.e., the click count). At last, the constructed KD-graph G has 629,875 keyword query nodes, 496,221 document nodes (URLs), and 2,778,050 edges. Dataset TWEET is based on 3,198,266 tweets published inside the New York area, collected using Twitter's Streaming API[7]. Each tweet has a text message and a location where the tweet is posted. Following the methodology in [33], we extracted phrases of length 1 to 10 from the text messages and used them to model the keyword queries. Only the phrases ending with either a noun or an adjective and with frequency at least 3 are kept, in order to reduce the number of noisy queries. We add an edge between a keyword query node k_i and a tweet node d_j if d_j contains keywords k_i. The edge weight is the tf-idf score of k_i w.r.t. d_j. Afterwards, documents with no connected keyword query nodes are removed. Finally, the constructed KD-graph has 781,465 keyword queries, 1,482,064 documents (tweets), and 12,078,958 edges.

From each dataset, we randomly selected 100 keyword queries and used them as a workload of user queries for our experiments. For each keyword query, the user location is considered to be the location of a randomly selected document from those that contain the keywords. The reasoning behind this is that the query location distribution typically tends to follow the distribution of the documents that contain the query.

All tested methods were implemented using Java. The experiments were run on a machine with Intel Core i7-3770 3.40GHz and 16GB main memory. Our LKS framework and the algorithms are evaluated under various parameter settings, as shown in Table 1. For each parameter, we used a wide range of values in order to test their effects. The number of suggestions m is fixed to 5 (the performance of the evaluated algorithms is not sensitive to it). Finally, the

6. http://freegeoip.net

7. https://dev.twitter.com/docs/api/streaming

TABLE 1
Parameters

Description	Parameter	Values	Default Value
Number of partitions	N	1,4,16,64,256	16
RWR probability	α	0.2,0.35,0.5,0.65,0.8	0.5
Edge weight adjustment param.	β	0,0.25,0.5,0.75,1	0.5
Threshold ($\times 10^{-5}$)	ϵ	0.1,0.5,1,5,10	1
Number of documents (M)	$\|D\|$	0.5, 1, 1.5, 2, 2.5	1.5

default partitioning method for PA is Spatial Partitioning.

4.2 Effectiveness of the LKS framework

In this section, we assess the effectiveness of LKS using both real examples and quantitative measurements, verifying whether the keyword suggestions are (i) semantically relevant to the original query and (ii) able to find documents that are close to the query location.

Case Study. We first demonstrate the effectiveness of considering the user location in query suggestion, by showcasing query examples on the TWEET data set. Consider a user-supplied keyword query with k_q = "*airport*". Figures 7 and 8 show the top-1 suggestion for k_q considering two different query locations λ_q and using various values of β. In all plots, the triangle denotes the query location, the crosses denote the *processed documents* by the PA algorithm (i.e., the documents that distribute ink during the course of the algorithm), and the circles are the retrieved document by the suggested keywords close to the query location (located within 10km from the query location). When $\beta = 1$, LKS ignores the query location; thus, no matter where the query location is, the suggestion for the user supplied keyword "*airport*" is always "*LaGuardia Airport*" (the most relevant phrase to "*airport*" when disregarding location data) as shown in Figures 7(a) and 8(a). When the query location is close to LaGuardia Airport, the suggested keywords "*LaGuardia Airport*" can retrieve some relevant documents nearby (the circles in Figure 7(a)). However, when the query location is close to Newark, there are no relevant documents around related to the suggestion (no circle in Figure 8(a)).

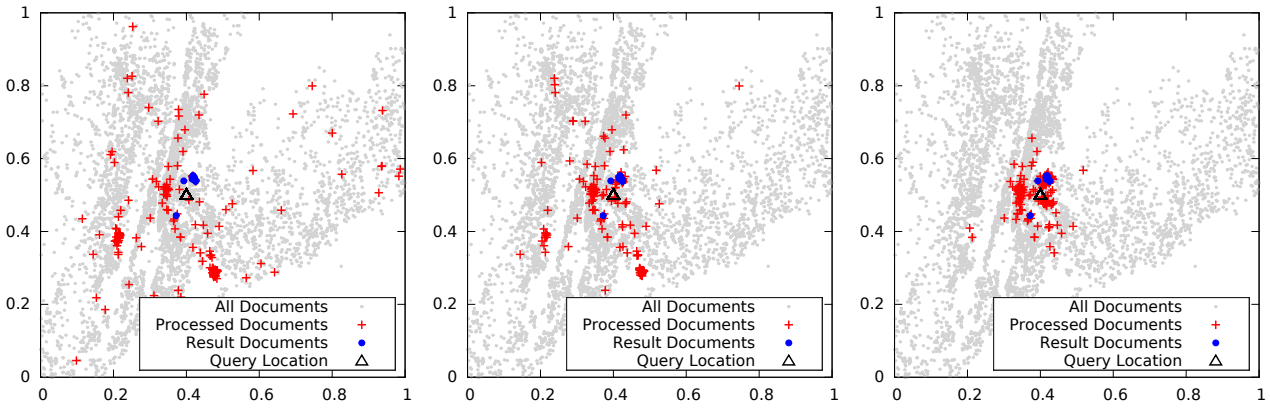
When $0 < \beta < 1$ ($\beta = 0.5$ in this example), the suggestion depends on the query location. Keywords "*LaGuardia Airport*" are suggested when the query is issued near LaGuardia (Figure 7(b)), while "*Newark Liberty International Airport*" is the top suggestion when the query location is close to Newark (Figure 8(b)). Each of the two suggestions can retrieve relevant documents (circles) close to the corresponding query location. When $\beta = 0$, the suggestion only depends on the query location and can fetch relevant documents nearby as shown in Figures 7(c) and 8(c). However, sometimes the suggestion is less specific (i.e., less relevant) compared to the case of $\beta = 0.5$, e.g., "*Newark Liberty International Airport*" vs. "*Newark Liberty*". This is because the spatial-only recommendation favors the queries having more relevant documents close to the query location, regardless of their textual relevance. When the query location is near JFK, "*John F Kennedy International Airport*" and "*Kennedy International*" are reported when β is set to 0.5 and 1.0, respectively. We omit the details due to space constraints.

We also observe that for small β values, fewer documents are involved in the computation (crosses) and only the nearby documents are considered. For example, the number of crosses is 2296, 2177, and 822 in Figures 8(a), 8(b), and 8(c), respectively. Moreover, small β values result in more general query suggestions, which retrieve more nearby documents. As an example, the number of circles is 600 and 630 in Figures 8(b) and 8(c), respectively.

A case where LKS is especially useful is when there are no relevant documents near the user location. LKS recommends to the user alternative query keywords, which match the user's intention and at the same time find nearby documents. For example, on our AOL dataset the query "*louis vuitton*" is recommended when an original query "*prada*" is issued at the place where no document is retrieved nearby by "*prada*". Suggestion "*louis vuitton*" succeeds in finding some relevant documents close to the user location.

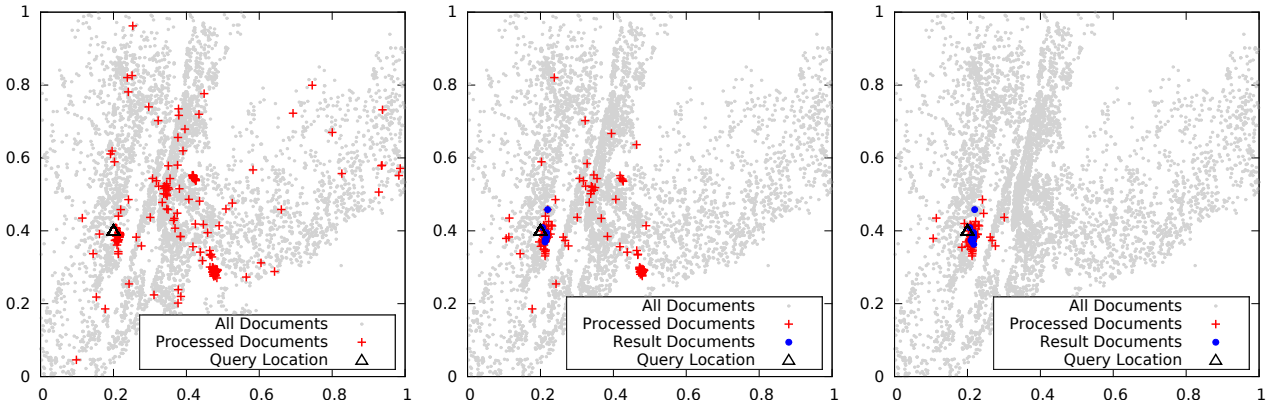
Quantitative Study. Designing quantitative evaluation metrics for LKS is challenging, because it is hard to establish the ground truth for suggestions and an online system for user evaluation is not yet available. Considering the two criteria of good suggestions, we evaluate (i) the semantic relevance of the suggested keywords (together with their retrieved nearby documents) w.r.t. the user's initial query and (ii) the number of nearby documents retrieved by the query suggestions. As a competitor to our LKS framework, we implemented the *influence tag co-occurrence* (INF) method proposed in [21]. INF is designed to employ both spatial and textual information into tag recommendation for Flickr and has been shown to be more effective than alternative approaches [21]. Given a photo p published at $p.loc$ and a tag t, the task is to retrieve k tag recommendations to t. The idea is that if a tag t' co-occurs with t in a photo p', they are considered textually relevant. In the meantime, if p' is close to p (i.e., $dist(p'.loc, p.loc)$ is small), the relevance between t' and t is even higher. In INF, the relevance between t and t' is defined according to a influence score $inf(t,t') = \frac{\sum_{p' \in P \cap P'} 2^{-\frac{-dist(p',p)}{r}}}{|P \cup P'|}$. Here P and P' are the sets of photos containing tag t and t', respectively, and r is a parameter to adjust the importance of spatial distance. To apply INF in keyword suggestion, we consider the co-occurrence of keywords in the same document and exploit a similar relevance function. We tested various r values from 0.1 to 0.0001 and observed that with decreasing r, more nearby documents are retrieved, but the semantic relevance of the suggested keywords declines. Thus, in the following, we present only the results with $r = 0.001$, which in our experiments retains the best balance between the two factors.

We first report the average number of nearby documents retrieved by (a) the original query, (b) the queries suggested by INF, and (c) the queries suggested by our LKS framework, over the workload of TWEET when varying parameters α (in Equation 3) and β (in Equation 1). Figure 9(a) plots the number of documents retrieved by the original input and the keyword queries suggested by INF or LKS, within a ratio ρ of the maximum Euclidean distance between any pair of documents ($\rho = \{0.05, 0.1\}$). The queries suggested by INF can retrieve some more nearby locations. However, the number of documents retrieved by the LKS-suggested



(a) β=1, top-1: *"LaGuardia Airport"* (b) β=0.5, top-1: *"LaGuardia Airport"* (c) β=0, top-1: *"LaGuardia Airport"*

Fig. 7. User Supplied Keyword k_q = *"airport"*, Query Location λ_q near LaGuardia Airport

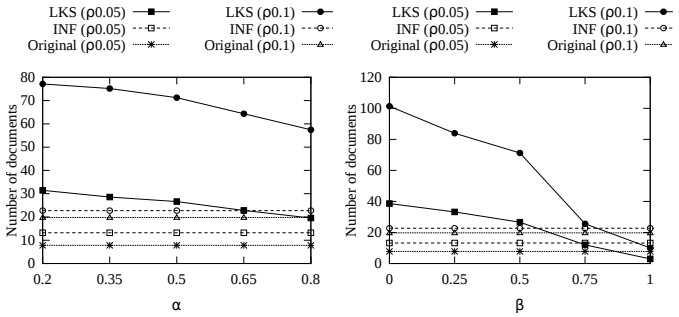


(a) β=1, top-1: *"LaGuardia Airport"* (b) β=0.5, top-1: *"Newark Liberty International Airport"* (c) β=0, top-1: *"Newark Liberty"*

Fig. 8. User Supplied Keyword k_q = *"airport"*, Query Location λ_q near Newark Liberty International Airport

queries is significantly higher than that of either the original input, or the INF suggested keyword queries, for all α values, showing the value of LKS w.r.t. the location criterion. Obviously, for larger ρ values more documents are retrieved. Also, the number of documents decreases as α increases. The reason is that for small α values, the distributed ink reaches more documents compared to the case where α is large. Hence, more document nodes near the query location participate in the ink distribution process and more relevant queries are reached and included in the top-m ranking. As special cases, when $\alpha = 1$, no keyword queries can be returned because all the ink is retained by the original query, while when $\alpha = 0$, no restart is involved in the random walk, hence the result is independent of the input. Figure 9(b) reports the number of documents retrieved by the original input and the keyword queries suggested by INF or LKS when varying β. The number of documents retrieved by the queries suggested by LKS decreases as β increases. This is because a large β weighs the user location low; the textual relevance criterion dominates the spatial closeness criterion, therefore fewer nearby documents are relevant to the suggestions. Note that our experiments on AOL show similar results and are thus omitted.

Next, we evaluate how relevant the suggested keyword queries are to the user's information needs. To begin with,



(a) Varying α (b) Varying β

Fig. 9. Number of Retrieved nearby Documents

we first evaluate the textual relevance between the original input and the keyword queries suggested by INF or LKS, following a similar approach used in [34], [35], [36]. Specifically, we utilize the Open Directory Project database (ODP, a.k.a. DMOZ)[8], which provides a comprehensive human edited directory of the Web. ODP can match a given query to a hierarchy of categories. For example, the query "Hard Disk" matches category "Computers: Hardware: Storage:

8. http://www.dmoz.org

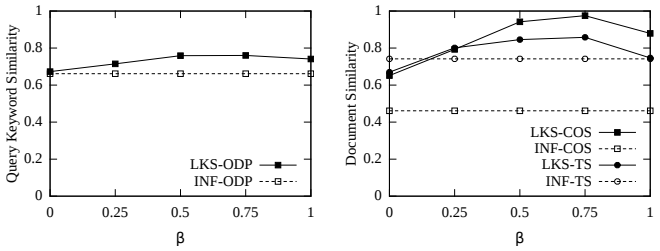
Hard Disk" and the query "Memory" matches "Computers: Hardware: Components: Memory". Hence, the relevance between two queries can be evaluated by the similarity between their corresponding categories C_1 and C_2 in the hierarchy. Similarity is measured by the length of the longest common path prefix $P(C_1, C_2)$ normalized by the length of the longer path in the hierarchy, i.e., $sim(C_1, C_2) = \frac{|P(C_1,C_2)|}{\max\{|C_1|,|C_2|\}}$. For example, the similarity between "Hard Disk" and "Memory" is $2/4$ as the longest common path "Computers: Hardware" has length 2. In our experiments, we evaluated the relevance between the original input and the top suggested keyword queries by INF or LKS, based on their most similar categories provided by ODP. The results are presented in Figure 10(a). The first observation is that the LKS-based suggestions are generally more promising than INF based ones, unless β is close to 0 (spatial only). This is because INF relies merely on co-occurrence to judge textual relevance, which cannot capture many hidden semantic relationships between keyword queries. LKS, on the other hand, captures textual proximity based on the whole structure of KD-graph and thus retrieves more reliable suggestions. In addition, we also observe that LKS achieves best ODP similarity when β is selected between 0.25 and 0.75. Naturally, when β is small (e.g. $\beta \to 0$), the graph is over-biased to spatial closeness and cannot reflect textual relevance clearly. On the other hand, when β is large (e.g., $\beta \to 1$), the graph does not benefit from the local intent of the input query.

After the direct evaluation of suggested query keywords in the previous experiment, we now evaluate the nearby documents retrieved by them. Let $\{d_i^o\}$ be the set of top-10 documents retrieved by the original query and $\{d_i^s\}$ be the set of top-10 documents retrieved by a suggested query within a spatial range ($\rho = 0.1$), where i indicates the ranking position of a document. We compute the cosine similarity $\cos(d_i^o, d_i^s)$ between the documents with the same ranking position from the two sets $\{d_i^o\}$ and $\{d_i^s\}$. We adopt the nDCG measure of [37] to aggregate these cosine similarities so that the document similarity is reduced logarithmically, proportionally to its ranking position.

$$COS = \frac{\cos(d_1^o, d_1^s) + \sum_{i=2}^{10} \cos(d_i^o, d_i^s)/log(i)}{1 + \sum_{i=2}^{10} 1/log(i)}.$$

The higher the value of COS is, the closer the nearby documents are to user's information needs. Besides, we also compare the textual relevance (i.e., $tf \times idf$) of documents $\{d_i^s\}$ and $\{d_i^o\}$ w.r.t. the original query. Let s_i^o and s_i^s be the textual score of document d_i^o and d_i^s w.r.t. the original query, respectively. We take the ratio of s_i^s to s_i^o for the same ranking position. Similar to COS, these ratios are aggregated as TS. Figure 10(b) shows the average COS and TS over the workload of TWEET when varying β. The results are mostly consistent with Figure 10(a). LKS again has better textual quality than INF unless β is very small (< 0.25). In the meantime, LKS achieves best effectiveness at non-extreme β values, where it takes advantage of both the textual relationships from the original KD-graph, and the local intent of the input query.

Considering the above factors, we conclude that an average value of β (e.g., $0.25 < \beta < 0.75$) should be used in order for the keyword query suggestions to have a sufficient number of documents near the user location as results, which also meet the information needs of the user. In summary, our LKS framework, which balances between relevant and nearby search results, is very effective, based on our experimental analysis.



(a) Keyword query similarity (b) Document similarity

Fig. 10. Textual Relevance Evaluation

4.3 Efficiency

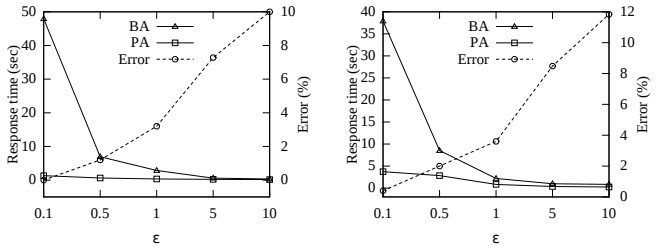
In this section, we evaluate the performance of BA and PA on the two data sets under various parameter settings and report their average runtime (over the workload of 100 queries[9]).

Varying ϵ. Both BA and PA follow the general idea of BCA [28], and their runtime depends on parameter ϵ. The smaller ϵ is, the smaller the error incurred by BCA (and BA/PA) on computing the PageRank scores on G_q. We find that when $\epsilon \leq 10^{-8}$, the top query suggestions become stable. Thus, we consider the top suggestions calculated using $\epsilon = 10^{-8}$ as the base for computing the approximation error of the top suggestions calculated using $\epsilon > 10^{-8}$. Following [27], the error is computed as $1 - AP$ where AP is the average precision of the retrieved results compared to the true ones. An appropriate ϵ value should be selected so that (1) the computation can be finished in real time and (2) the error rate is reasonably low. Figure 11 shows that the response time of BA drops dramatically as ϵ increases, since a large ϵ significantly reduces the number of iterations in BA and saves many computations. However, the approximation error is high for large values of ϵ. On the other hand, PA runs fast even for small values of ϵ, for which the approximation error is low. PA outperforms BA by 1-2 orders of magnitude when ϵ is smaller than 10^{-6}. Therefore, under the same time constraint (e.g. $<$ 1s), PA can support smaller ϵ values, and can thus guarantee a lower error rate.

Because of ϵ, BA and PA may produce different suggestions, but the differences are mainly in the low-ranked suggestions. In practice, users only consider the highly ranked suggestions. In our experiments, the top-5 suggestions offered by BA and PA are identical in 98% and 99% of the times on AOL and TWEET, respectively.

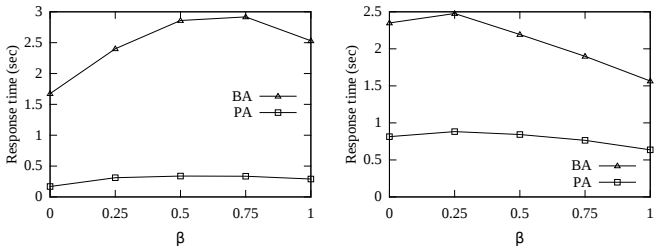
Varying α. Parameter α in Equation 3 indicates the random walk restart probability that corresponds to the proportion of ink retained on a query node. As shown in Figure 12 the response times of both BA and PA decrease as α increases. This is because for larger α, more amount of ink

9. Experiments with larger workloads gave us similar results.



(a) AOL (b) TWEET

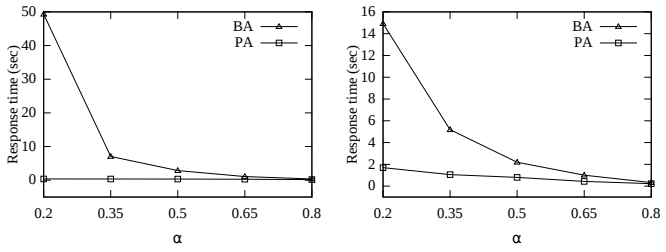
Fig. 11. Response Time when Varying ϵ ($\times 10^{-5}$)



(a) AOL (b) TWEET

Fig. 13. Response Time when Varying β

is retained at the keyword query nodes; thus, the amount of active ink drops quickly and the termination condition is satisfied early. We observe that PA is more robust to α and outperforms BA significantly when α is small. In the special case, when $\alpha = 1$, all the ink will be retained on the query node in the beginning, thus both algorithms terminate immediately returning no results. In the other extreme case, when $\alpha = 0$, no ink is retained in any node at each step, therefore the ink keeps being redistributed until the random walk process converges to a stable state. In this case, the final scores of the nodes only depend on the structure of the graph, and not on the starting node (query node); no matter what query is given, the suggestions are always the same (i.e., similar to global PageRank scores). Therefore, both extreme cases ($\alpha = 1$ or 0) cannot give useful results and are not considered.

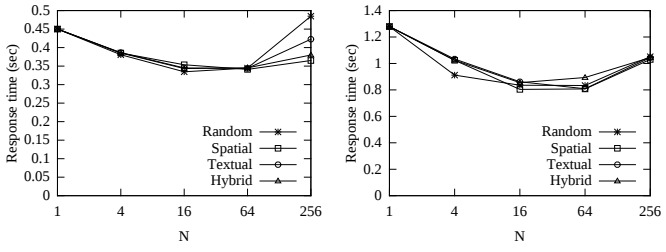


(a) AOL (b) TWEET

Fig. 12. Response Time when Varying α

Varying β. Parameter β in Figure 13 shows the response time when varying β. Both BA and PA are bell-shaped. When β approaches 0 or 1, only the documents that are either close to the query location or have high textual relevance to the user supplied keywords are involved in the computation. However, when β is close to 0.5, the response time increases slightly, since more documents are considered, i.e., the ones that are a bit far from the query location but with high textual relevance and the ones that are close to the query location but with slightly low textual relevance. Algorithm PA outperforms BA for all values of β by a wide margin.

Varying Partitioning Methods and Number of Partitions. We evaluate the four partitioning methods introduced in Section 3.2 using various number of partitions on the two datasets. The result is shown in Figure 14. The performance of PA seems to be insensitive to which partitioning approach is used. This can be attributed to the fact that the KD-graph and its corresponding matrices are very sparse. Therefore, the difference between the clusters formed by these approaches is not significant enough to influence the performance of PA. In the future, we plan to study the applicability of PA on otherwise distributed scenarios (where transition matrices are denser). The number of partitions N, on the other hand, directly affects the performance. PA performs best when the number of partitions is between 16 and 64. This is because when N is small, e.g., 1 and 4, the lazy distribution mechanism has little effect, since the size of each partition is large and there is little chance that the ink is being accumulated. Thus, the cost saved by the ink accumulation is limited. In the special case $N = 1$ (a single keyword / document partition), every keyword / document node distribute their ink regardless of their remaining ink (i.e., there is no priority order). On the other hand, when N is large (e.g., 256), the lazy distribution mechanism is applied for too many partitions and the bookkeeping overhead at each node for the accumulated ink per partition is high, negatively affecting the performance of PA and counterbalancing the savings due to the lazy distribution. As a special case when $N = \max\{|K|, |D|\}$ (i.e., each keyword / document itself forms a partition), the PA algorithm falls back to BA.



(a) AOL (b) TWEET

Fig. 14. Varying Partitioning Methods and N

Varying dataset size. To test the effect of data size, we constructed a large TWEET_{US} dataset from about 10M tweets published in general U.S. area, in similar way to the TWEET dataset. We use subsets of TWEET_{US} with various numbers of documents / keywords. Figure 15 reports the response time of BA and PA when $|D|$ varies from 0.5M to 2.5M. Note that with different $|D|$, the corresponding keyword node set size $|K|$ also changes, as shown on the top of Figure 15. Based on the experiments, PA constantly outperforms BA by 3-4 times with various $|D|$ or $|K|$ settings.

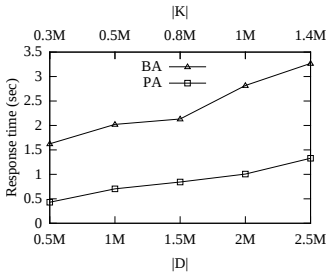


Fig. 15. Varying Number of Documents and Keywords

5 Related Work

Related work on query suggestion is discussed in Section 5.1. Techniques for RWR computation are reviewed in Section 5.2.

5.1 Keyword Query Suggestion

Keyword query suggestion approaches can be classified into three main categories: random walk based approaches, learning to rank approaches, and clustering based approaches. We also briefly review alternative methods that do not belong to any of these categories. To the best of our knowledge, no previous work considers user location in query suggestion.

Random Walk based Approaches. The methods in this category use a graph structure to model the information provided by query logs, and then apply a random walk process on the graph to compute the suggestions. Craswell and Szummer [5] apply such an approach on the query-click graph and suggest queries based on personalized PageRank scores. Mei et al. [6] rank queries based on hitting time, which reflects the probability that a random walker arrives a node within certain steps. Song et al. [7] combine both the clicked and skipped URLs from users in the query-URL bipartite graphs in order to also consider rare query suggestions (using clicked URLs only favors popular queries). This work was extended in [38] to achieve diversification for the recommended queries; the suggested queries are re-ranked in a way that maximizes the diversification function. Zhu et al. [39] generate query recommendations by considering diversity and relevance in a unified way during RWR search. Miyanishi et al. [8] introduce Time-aware Structured Query Suggestion (TaSQS), an algorithm for presenting query suggestions along a timeline. Our proposed LKS framework is orthogonal to and can be easily integrated in all the suggestion methods that use the query-URL bipartite graph.

Boldi et al. [10] proposed an approach that applies RWR from the original query or from a small set of queries representing the recent querying history of the current user on a *query-flow* graph. Deng et al. [36] investigate and develop an entropy-biased framework for modeling click graphs, in which common clicks on frequent and general pages are given less weight than the ones on less frequent but more specific pages. Anagnostopoulos et al. [9] formulate the query-recommendation problem as a decision problem on how to perturb the transition probabilities between queries in the query-flow graph so as to maximize the expected utility of a random walk. Song et al. [11] mine a term-transition graph from search engine logs and apply a topic-based unsupervised Pagerank model that suggests queries based on the topic distribution and term-transition probability within each topic. The idea of our LKS framework can be applied to methods that use a query-flow graph, since the query history may also have location information. This is an interesting subject for our future work.

Learning to Rank Approaches. Some query suggestion approaches [13] are based on learning models trained from co-occurrences of queries in search logs. Another learning-to-rank approach [40] is trained based on several types of query features, including query performance prediction. Li et al. [12] train a hidden topic model. For each candidate query, its posterior distribution over the hidden topic space is determined. Given a user query q, a list of suggestions is produced based on their similarity to q in the topic distribution space. Our work is not based on learning models; in the future, it would be interesting to study how these models can be extended to consider location information.

Clustering based Approaches. Beeferman and Berger [3] view the query log as a query-URL bipartite graph. By applying an agglomerative clustering algorithm on the vertices in the graph, query clusters can be identified. Then, given a user-supplied query q, the queries that belong to the same cluster as q does are returned to the user as suggestions. [41] further extended the approach to also take into account the similarity between the query content during clustering. In [2], a similar approach is proposed: the queries are modeled as term-weighted vectors and then clustered. The vector of a query q includes the clicked URLs by the users who posed q as terms and the weights are calculated based on term frequency and the click popularity of the URL in the answers of q. Cao et al. [4] take into account the immediately preceding queries as context in query suggestion. They summarize queries in search logs into concepts by clustering a query-URL bipartite graph. User session data are converted to concept sequences and indexed by a suffix tree. The query sequence submitted by the user is mapped to a sequence of concepts; the suffix tree is then searched to find query suggestions. Finally, Li et al. [42] cluster queries from search logs to extract query concepts, based on which recommended queries are selected and employ a probabilistic model and a greedy heuristic algorithm to achieve recommendation diversification. Location information could also be considered in all these clustering models. Such an approach is out of the scope of our current work, but we are interested in investigating its effectiveness in the future.

Miscellaneous Approaches. Zhang and Nasraoui [43] try to create a graph with edges between consecutive queries in each session, weighted by the textual similarity between these queries. A candidate suggestion for a given query is given a score based on the length of the path between the two queries, aggregated across all sessions in a query log where the query and the suggestion co-occurred. Cucerzan and White [44] propose to generate query suggestions based on user *landing pages* (that is, the web pages that users end a query with, through post-query browsing). Given a user query, they utilize its recorded landing pages and suggest to the user other queries that have these landing pages

in their top ranked results. A probabilistic mechanism [33] generates query suggestions from the corpus without using query logs.

Location-aware type-ahead search. References [19] and [20] both study the problem of location-aware type-ahead search (LTAS), also known as *instant search*. LTAS finds documents near a user location, as the user types in a keyword query character by character. This problem is more related to keyword query completion than to the query suggestion problem that we study in this paper, since the recommended keywords must have the user's input as prefix. On the other hand, the query suggestion problem that we study in this paper takes a completed query and recommends other queries that are semantically relevant without the constraint that the suggestions should have the original user query as prefix. Therefore, our LKS framework is more flexible and can help users to express various aspects of a topic. The suggested keywords can be different than the user-supplied keywords, but they should be textually relevant. In addition, the methods for LTAS are very different to our LKS algorithms, as they take advantage of the prefix requirement to reduce the search space (with the help of trie data structures).

Location-aware suggestions based on user history. Google [45] provides location-based query suggestions by simply selecting the user's past search queries that have results close to the user's current location. These suggestions may be insufficient if the user did not perform any historical searches near her current location. In addition, query suggestion based on location only may not match the user's search intent. On the other hand, our framework aims at suggesting keyword queries that satisfy the user's information needs and have nearby results, irrespectively to the user's search history.

Query relaxation. The database research community has studied a relevant problem to query suggestion, called *query relaxation*. The objective is to generalize an SQL query in case it returns too few or no results [46]. Query relaxation approaches cannot be applied for keyword query suggestion, because they require the relaxed query to contain the results of the original query, which is not essentially the case in query suggestion.

5.2 Random Walk Computation

Random walk with restart (RWR), also known as Personalized PageRank (PPR), has been widely used for node similarity measures in graph data, especially since its successful application by the Google search engine [47].

Pre-computation based Approaches. Some matrix-based methods [22], [23] solve PPR by pre-computing the inversion matrix. Tong et al. [22] propose a matrix-based approach B_LIN that reduces the pre-computation cost of the full matrix inversion by partitioning the graph. Fujiwara et al. [23] propose a K-dash method that finds the top-k nodes with the highest PPR scores, based on a LU decomposition of the transition matrix. Alternative to matrix-based approaches, Monte Carlo (MC) methods [24], [25], [26] can be used to simulate the RWR process. Fogaras et al. [24] propose to approximate PPR by pre-computing and approximating for each node u a set of ending vertices for random walks starting from u. If u later becomes a query node, its PPR is approximated according to the distribution of these vertices. Similarly, Bahmani et al. [25] approximate PPR by counting the number of times a node is visited by pre-computed random walk paths. All above methods require the apriori knowledge of the complete graph; however, in our problem, the edge weights are dynamically adjusted according to the user location, thus these approaches are inapplicable.

Online Approaches. MC can also be applied online, without relying on pre-computations; a number of random walks are tried from the query node and the PPR score of other nodes are estimated from these samples [26]. However, as shown later in [27], a large number of (expensive) random walks are required in order to achieve acceptable precision. Fujiwara et al. [27] propose a method for efficient ad-hoc top-k PPR search with exact node ranking. They compute the random walk (without restart) probabilities of the nodes, and employ the probabilities to estimate upper/lower bounds of the candidate nodes. This approach is applicable when the complete transition matrix is available beforehand, however, obtaining the complete transition matrix in our problem involves the multiplication of two matrices and it is very expensive. Berkhin [28] proposes the Bookmark Coloring Algorithm (BCA) to derive an approximation of the PPR vector. Gupta et al. [30] extend BCA and employ early termination heuristics for the top-k PPR calculation. We extend algorithm BCA [28] as our baseline algorithm (BA) to compute the location-aware suggestions.

6 Conclusion

In this paper, we proposed an LKS framework providing keyword suggestions that are relevant to the user information needs and at the same time can retrieve relevant documents near the user location. A baseline algorithm extended from algorithm BCA [28] is introduced to solve the problem. Then, we proposed a partition-based algorithm (PA) which computes the scores of the candidate keyword queries at the partition level and utilizes a lazy mechanism to greatly reduce the computational cost. Empirical studies are conducted to study the effectiveness of our LKS framework and the performance of the proposed algorithms. The result shows that the framework can offer useful suggestions and that PA outperforms the baseline algorithm significantly. In the future, we plan to further study the effectiveness of the LKS framework by collecting more data and designing a benchmark. In addition, subject to the availability of data, we will adapt and test LKS for the case where the locations of the query issuers are available in the query log. In addition, we believe that PA can also be applied to accelerate RWR on general graphs with dynamic edge weights and we will investigate its general applicability in the future. Moreover, the current version of PA seems to be independent of the partitioning method. It would be interesting to investigate whether alternative partitioning heuristics can further reduce the cost of the algorithm.

References

[1] M. P. Kato, T. Sakai, and K. Tanaka, "When do people use query suggestion? A query suggestion log analysis," *Inf. Retr.*, vol. 16, no. 6, pp. 725–746, 2013.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Query recommendation using query logs in search engines," in *EDBT*, 2004, pp. 588–596.
[3] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *KDD*, 2000, pp. 407–416.
[4] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li, "Context-aware query suggestion by mining click-through and session data," in *KDD*, 2008, pp. 875–883.
[5] N. Craswell and M. Szummer, "Random walks on the click graph," in *SIGIR*, 2007, pp. 239–246.
[6] Q. Mei, D. Zhou, and K. Church, "Query suggestion using hitting time," in *CIKM*, 2008, pp. 469–478.
[7] Y. Song and L.-w. He, "Optimal rare query suggestion with implicit user feedback," in *WWW*, 2010, pp. 901–910.
[8] T. Miyanishi and T. Sakai, "Time-aware structured query suggestion," in *SIGIR*, 2013, pp. 809–812.
[9] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis, "An optimization framework for query recommendation," in *WSDM*, 2010, pp. 161–170.
[10] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna, "The query-flow graph: Model and applications," in *CIKM*, 2008, pp. 609–618.
[11] Y. Song, D. Zhou, and L.-w. He, "Query suggestion by constructing term-transition graphs," in *WSDM*, 2012, pp. 353–362.
[12] L. Li, G. Xu, Z. Yang, P. Dolog, Y. Zhang, and M. Kitsuregawa, "An efficient approach to suggesting topically related web queries using hidden topic model," *WWW*, pp. 273–297, 2013.
[13] U. Ozertem, O. Chapelle, P. Donmez, and E. Velipasaoglu, "Learning to suggest: A machine learning framework for ranking query suggestions," in *SIGIR*, 2012, pp. 25–34.
[14] 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, 2013.
[15] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *VLDB J.*, vol. 21, no. 6, pp. 797–822, 2012.
[16] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu, "SEAL: Spatio-textual similarity search," *PVLDB*, vol. 5, no. 9, pp. 824–835, 2012.
[17] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-textual similarity joins," *PVLDB*, vol. 6, no. 1, pp. 1–12, 2012.
[18] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi, "Efficient algorithms and cost models for reverse spatial-keyword *k*-nearest neighbor search," *ACM Trans. Database Syst.*, vol. 39, no. 2, 2014.
[19] S. Basu Roy and K. Chakrabarti, "Location-aware type ahead search on spatial databases: Semantics and efficiency," in *SIGMOD*, 2011, pp. 361–372.
[20] R. Zhong, J. Fan, G. Li, K.-L. Tan, and L. Zhou, "Location-aware instant search," in *CIKM*, 2012, pp. 385–394.
[21] I. Miliou and A. Vlachou, "Location-aware tag recommendations for flickr," in *DEXA*, 2014, pp. 97–104.
[22] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *ICDM*, 2006, pp. 613–622.
[23] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-*k* search for random walk with restart," *PVLDB*, vol. 5, no. 5, Jan. 2012.
[24] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," *Internet Math*, vol. 2, no. 3, pp. 333–358, 2005.
[25] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," *PVLDB*, vol. 4, no. 3, pp. 173–184, Dec. 2010.
[26] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol, "Quick detection of top-*k* personalized pagerank lists," in *Algorithms and Models for the Web Graph*, 2011, vol. 6732, pp. 50–61.
[27] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka, "Efficient ad-hoc search for personalized pagerank," in *SIGMOD*, 2013, pp. 445–456.
[28] P. Berkhin, "Bookmark-coloring algorithm for personalized pagerank computing," *Internet Math*, vol. 3, pp. 41–62, 2006.
[29] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003, pp. 271–279.
[30] M. Gupta, A. Pathak, and S. Chakrabarti, "Fast algorithms for top*k* personalized pagerank queries," in *WWW*, 2008, pp. 1225–1226.
[31] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *KDD*, 2001, pp. 269–274.
[32] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *InfoScale*, 2006.
[33] S. Bhatia, D. Majumdar, and P. Mitra, "Query suggestions in the absence of query logs," in *SIGIR*, 2011, pp. 795–804.
[34] R. Baeza-Yates and A. Tiberi, "Extracting semantic relations from query logs," in *KDD*, 2007, pp. 76–85.
[35] H. Ma, H. Yang, I. King, and M. R. Lyu, "Learning latent semantic relations from clickthrough data for query suggestion," in *CIKM*, 2008, pp. 709–718.
[36] H. Deng, I. King, and M. R. Lyu, "Entropy-biased models for query representation on the click graph," in *SIGIR*, 2009, pp. 339–346.
[37] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, Oct. 2002.
[38] Y. Song, D. Zhou, and L.-w. He, "Post-ranking query suggestion by diversifying search results," in *SIGIR*, 2011, pp. 815–824.
[39] X. Zhu, J. Guo, X. Cheng, P. Du, and H.-W. Shen, "A unified framework for recommending diverse and relevant queries," in *WWW*, 2011, pp. 37–46.
[40] Y. Liu, R. Song, Y. Chen, J.-Y. Nie, and J.-R. Wen, "Adaptive query suggestion for difficult queries," in *SIGIR*, 2012, pp. 15–24.
[41] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang, "Clustering user queries of a search engine," in *WWW*, 2001, pp. 162–168.
[42] R. Li, B. Kao, B. Bi, R. Cheng, and E. Lo, "DQR: A probabilistic approach to diversified query recommendation," in *CIKM*, 2012, pp. 16–25.
[43] Z. Zhang and O. Nasraoui, "Mining search engine query logs for query recommendation," in *WWW*, 2006, pp. 1039–1040.
[44] S. Cucerzan and R. W. White, "Query suggestion based on user landing pages," in *SIGIR*, 2007, pp. 875–876.
[45] J. Myllymaki, D. Singleton, A. Cutter, M. Lewis, and S. Eblen, "Location based query suggestion," Oct. 30 2012, uS Patent 8,301,639.
[46] T. Gaasterland, "Cooperative answering through controlled query relaxation," *IEEE Expert*, vol. 12, no. 5, pp. 48–59, Sep 1997.
[47] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Comput. Netw. ISDN Syst.*, vol. 30, no. 1-7, pp. 107–117, Apr. 1998.