International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 07
June 2017

Location Aware Keyword Query Suggestion

1. P DIVYA, 2.DR. K NAGESWARARAO

[1]Pg Scholar, Department of CSE, Mother Teresa Institute of Science and Technology, Sathupally,
divya.potlapalli@gmail.com

[2] Professor & HOD, Department of CSE, Mother Teresa Institute of Science and Technology, Sathupally,
nageswararaokapu@yahoo.com

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 cooccurrences in query sessions [13], and (iv) based on their similarity in the topic distribution space [12]. However, none of the existing methods provide locationaware 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 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 20111 , 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 • 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 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 d1–d5 as listed in

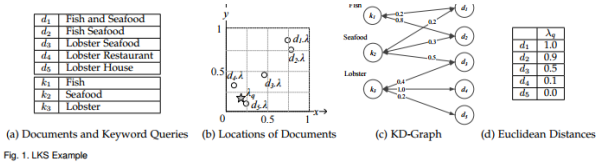

Fig. 1. LKS Example

Figure 1(a). Each document di is associated with a location di .λ as shown in Figure 1(b). Assume that a user issues a keyword query kq = "seafood" at location λq, shown in Figure 1(b). Note that the relevant documents d1–d3 (containing "seafood") are far from λq. A locationaware suggestion is "lobster", which can retrieve nearby documents d4 and d5 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 tag recommendation).

3 LKS FRAMEWORK

Consider a user-supplied query q with initial input kq; kq can be a single word or a phrase. Assuming that the query issuer is at location λq, 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 kq and (ii) the suggested queries can retrieve relevant documents spatially close to λq. The proposed LKS framework captures these two criteria. 2.1 Keyword-Document Graph Without loss of generality, we consider a set of geodocuments D such that each document di ∈ D has a point location di .λ. 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 di is clicked by a user who issued keyword query kj in the query log, E contains an edge e from kj to di and an edge e 0 from di to kj . Initially, the weights of edges e and e 0 are the same and equal to the number of clicks on document di , given keyword query kj [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. Locationindependent 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 d1–d5 and three keyword queries k1–k3. 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).

3.2 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 kq (a word or a phrase) and a query location λq. Given q, the weight w(e) of the edge e from a keyword query node ki to a document node dj is adjusted by the following function: $\tilde{w}(e) = \beta \times w(e) + (1 - \beta) \times (1 - dist(\lambda q, dj.\lambda))$ (1) where w(e) is the initial weight of e in the KD-graph, $\tilde{w}(e)$ is the adjusted edge weight, dist(λq, dj .λ) is the Euclidean distance between the query issuer's location λq and document dj , and parameter $\beta \in [0, 1]$ is used to balance the importance between the original (i.e., click-based) weight and the distance of dj to the query location. Euclidean distances are normalized to take values in [0, 1]. This keywordto-document edge weight adjustment increases the weights of the documents that are close to the user's location. Let D(ki) be the set of documents connected to a keyword query $ki \in K$ in the KD-graph. D(ki) may contain multiple documents and the locations of them form a spatial distribution. We propose to adjust the weights of the edges pointing to ki by the minimum distance between λq and the locations of documents in D(ki). 4 Such an adjustment favors keyword query nodes which have at least one relevant document close to the query issuer's location λq. Specifically, the weight w(e 0) of the edge e 0 from a document node dj to a keyword query node ki is adjusted as follows: $\tilde{w}(e\,0) = \beta \times w(e\,0) + (1 - \beta) \times (1 - mindist(\lambda q, D(ki)))$ (2) where mindist(λq, D(ki)) is the minimum Euclidean distance5 between λq and any document in D(ki). For example, Figure 1(b) shows the locations of the 5 documents of Figure 1(a) and a query location λq; Figure 1(d) includes the (approximate) Euclidean distances between λq and the five documents. Figure 2

illustrates how the edge 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(ki) is similar. 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 k1 to d1 as a concrete example. Its weight is calculated using Equation 1 where $dist(\lambda q, d1.\lambda) = 1$. The weight of the edge from d1 to k1 is computed using Equation 2 where $D(k1) = \{d1, d2\}$ and $mindist(\lambda q, D(k1)) = 0.9$. 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 KDgraph are independent to our edge weight adjustment strategy, which is query-dependent. Given a user-supplied query q, the adjusted graph Gq is dynamically derived from G based on the query location λq, used to compute suggestions for q, and then dropped. During this process, Gq 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

4 ALGORITHMS

In this section, we introduce a baseline algorithm (BA) to compute the location-aware suggestions. Then, we propose a more efficient partition-based algorithm (PA) (Baseline Algorithm (BA) We extend the popular Bookmark-Coloring Algorithm (BCA) 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 kq, BA processes the nodes in the graph in descending order of their active ink. Different from typical personalized PageRank problems where the graph is homogeneous, our KD-graph Gq 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−α 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 λq 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., Gq 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 . Algorithm BA terminates when either (i) the ink retained at the topmth 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 (typically, = 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 kq 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

ALGORITHM 1:

Baseline BA

Input : G(D, K, E), q = (kq, λq), m,

Output: C

1 PriorityQueue Q ← ∅, C ← ∅

2 Add kq to Q with kq.aink ← 1

3 AINK ← 1

4 while Q 6= ∅ and Q.top.aink ≥1 do

5 Deheap the first entry top from Q

6 tm = the top-m entry from C

7 tm0 = the top-(m + 1) entry from C

8 if tm.rink > tm0 .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 × w˜(top, v)

21 if there exists a copy v 0 of v in Q then

22 Remove v 0 from Q; v.aink ← v.aink + v 0 .aink

23 Add v to Q

24 return the top-m entries (excluding kq) in C involves retaining α portion of its active ink (line 13) and distributing 1 − α 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 kq in C as the result (line 24). Example 2. Figure 3 shows the steps of BA (for m = 1, = 0.1 and α = 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 k2, i.e., the keyword query kq = "seafood" supplied by the user. In the first iteration, node k2 retains 0.5 amount of ink and distributes 0.5 amount of ink to its neighbor document nodes d1–d3 according to the adjusted edge weights. In the second iteration, d3 distributes its active ink of amount 0.325 to its neighbor keyword query nodes k2 and k3. BA terminates at the sixth iteration where the active ink of each node is smaller than . The top-1 suggestion (excluding user query k2) is k3 = "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. 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) require complete knowledge of the graph before the algorithm starts. Therefore, they are not applicable to our problem, because the edge weights in graph Gq depend on the query location, which is unknown in advance. Applying a pre-computation technique for all possible query locations (i.e., all possible Gq) 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 P K = {P K i } be the partitions of the keyword queries and P D = {P D i } 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 GKP and GDP constructed offline from the KD-graph G and partitions P K and P D. In graph GKP , a keyword query node ki connects to a document partition P D if ki connects in G to at least one document in P D. Similarly, in graph GDP , a document node dj connects to a keyword partition P K if dj connects in G to at least one keyword query node ki . As an example, in Figure 4, the document partitions are P D 1 = {d1, d2} and P D 2 = {d3, d4, d5} and the keyword query partitions are P K 1 = {k1} and P K 2 = {k2, k3}. The edge weights are defined based on graph Gq, 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 Gq. (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 k2 0.35 0.65 d1 d2 d3 d4 k3 k1

1.0 1.0 PD 1 PD 2 PK 1 PK 2 d5 (a) Keywords to Partitions k2 d1 d2 d3 d4 k3 k1 PD 1 PD 2 PK 1 PK 2 d5 0.3 0.7 0.5 0.5 1.0 1.0 1.0 (b) Documents to Partitions. 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 k2 distributes its active ink to each of its three neighbor nodes d1–d3. However, in algorithm PA, the active ink of k2 is only distributed to two recipients: partitions P D 1 and P D 2 ; an underlying document node will not receive the ink, until its partition reaches the top of the queue. Lazy Distribution Mechanism.. This is an interesting subject for our future work. Learning to Rank Approaches. Some query suggestion approaches are based on learning models trained from co-occurrences of queries in search logs. Another learningto-rank approach 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. 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. 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 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 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 generates query suggestions from the corpus without using query logs. Location-aware type-ahead search. References and 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 usersupplied 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 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 . 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 precomputed 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 proposes the Bookmark Coloring Algorithm (BCA) to derive an approximation of the PPR vector. Gupta et al extend BCA and employ early termination heuristics for the top-k PPR calculation. We extend algorithm BCA 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 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. TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF HONG KONG 14

[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.