Noname manuscript No.
(will be inserted by the editor)

Location-Aware Query Reformulation for Search Engine

Zhipeng Huang[†] · **Yuqiu Qian**[†] · **Nikos Mamoulis**[‡]

the date of receipt and acceptance should be inserted later

Abstract Query reformulation, including query recommendation and query auto-completion, is a popular add-on feature of search engines, which provide related and helpful reformulations of a keyword query. Due to the dropping prices of smartphones and the increasing coverage and bandwidth of mobile networks, a large percentage of search engine queries are issued from mobile devices. This makes it possible to improve the quality of query recommendation and auto-copmpletion by considering the physical locations of the query issuers. However, limited research has been done on location-aware query reformulation for search engines. In this paper, we propose an effective spatial proximity measure between a query issuer and a query with a location distribution obtained from its clicked URLs in the query history. Based on this, we extend popular query recommendation and auto-completion approaches to our location-aware setting, which suggest query reformulations that are semantically relevant to the original query and give results that are spatially close to the query issuer. In addition, we extend the bookmark coloring algorithm for graph proximity search to support our proposed query recommendation approaches online, and we adapt an A* search algorithm to support our query auto-completion approach. We also propose a spatial partitioning based approximation that accelerates the computation of our proposed spatial proximity. We conduct experiments using a real query log, which show that our proposed approaches significantly outperform previous work in terms of quality, and they can be efficiently applied online.

[†]The University of Hong Kong
E-mail: {zphuang, yqqian}@cs.hku.hk
[‡]University of Ioannina
E-mail: nikos@cs.uoi.gr

1 Introduction

Keyword search is the standard tool of Web search engines, allowing users to express their search needs by typing a few keywords. Recently, there has been a lot of interest by both research and industry in the subject of *keyword query reformulation*. Besides ranking the Web pages according to their relevance to the keyword query provided by the user, a search engine may provide several alternative formulations of the query, which can be more focused and of interest to the user. Query reformulation has two types: *query recommendation* and *query auto-completion*. Query recommendation provides reformulations after the query has been issued, while query auto-completion suggests possible extensions to the query while it is being typed. Query reformulation, as an add-on function to a search engine, has been proved to greatly improve user experience [12].

Most of the existing work on query reformulation focuses on the analysis of *query logs*, which contain large amounts of historical information of search engine users, including what query was issued by whom at what time and which URLs were subsequently clicked [6, 7, 9, 18]. The query logs are often represented as *graphs* of queries and other related components, allowing graph analysis techniques to perform relevance search on query logs. For example, [6] built a graph of *queries* based on query logs. The weight of a graph edge that connects two queries is proportional to the times that the two queries are issued by the same user within a short period (i.e., the queries are in the same *search session*). Query recommendation is performed by applying Personalized PageRank proximity search on this graph starting from the original query. In another example, [8] builds a prefix-tree of queries, and query auto-completion is performed by applying A* search on the prefix-tree.

Nowadays, as mobile devices are ubiquitous, many keyword search queries are expressed by mobile users and have *spatial intent*, i.e., the users require results related to entities that are physically close to their locations. However, very few studies have considered location information when performing query reformulation. A recent piece of work [23] proposes a solution based on a bipartite graph that connects queries to their clicked documents (or URLs). The edges of the graph are adjusted based on the location of the query issuer and then Personalized PageRank proximity search is applied to obtain the recommendations. The work of [23] only considers the locations of documents to derive the proximity between queries. In this paper, we propose a more sophisticated approach that generates a spatial distribution for each query (based on its clicked URLs) and uses it to directly measure the proximity between each query and the query issuer.

The main technical challenge is efficiency; search engines should provide instant responses to users; hence, query reformulation should also be conducted in sub-seconds. However, differently from the traditional location-agnostic setting, which ignores user locations, we need to consider the spatial proximity between queries and the user, which can only be obtained online after the user issues her query. For query recommendation, we first adopt Bookmark Color-

ing Algorithm (BCA) [5], a classic method for online Personalized PageRank based proximity search, to support our recommendation method. Then, we design an *approximate spatial proximity* measure, which is based on a spatial partitioning and can be computed fast without sacrificing the quality of the results.

In a preliminary version of this paper [19], we studied location-aware query recommendation and extended two popular approaches, i.e., query-flow graph [6] and term-query-flow graph [7], to the location-aware setting. In this paper, we also consider the problem of location-aware query auto-completion, which aims at finding queries that are essentially extensions of the input query and are spatially close to the user. We formulate this problem by combining the original frequency-based ranking score and our location-based ranking score. We also propose an A* search algorithm that supports our proposed auto-completion model, providing query suggestions instantaneously.

We perform extensive experiments on a real query log from to verify the performance of our methods. We first conduct a user study, which shows that the users prefer recommendations generated by our proposed spatial proximity measure compared to four alternatives. Then, we compare our proposed location-aware query recommendation approaches with previous work, and show that our approaches achieve significantly better recommendations in terms of both semantic relevance and spatial proximity. Our experiments on efficiency show that our proposed algorithms on both recommendation and auto-completion can provide online query reformulations.

The contributions of this paper are summarized as follows:

- We consider the spatial proximity between a query and a search engine user in location-aware query reformulation. We propose an effective spatial proximity measure, shown to outperform alternatives in our user study.
- We extend two popular query recommendation approaches to support location-aware recommendation.
- We extend an A* search algorithm to support location-aware query auto-completion.
- We evaluate our proposed methods, demonstrating that our methods outperform previous work significantly and that they can suggest query reformulations instantaneously.

The rest of the paper is organized as follows. Section 2 presents definitions and background on the problem. Sections 3 and Section 4 introduce our location-aware query auto-completion model and our location-aware query recommendation model, respectively. Section 5 outlines our practical, approximate spatial proximity measure. Section 6 includes our experimental results. Section 7 reviews related work and Section 8 concludes the paper.

2 Preliminaries and Definitions

In this section, we introduce some necessary preliminaries and definitions, including *query logs* (Section 2.1), how we model and obtain the relevance

of queries to locations (Section 2.2), and two popular location-agnostic query recommendation methods (Sections 2.3 and 2.4). Then we introduce prefix-tree, a commonly-used data structure for efficient query auto-completion in Section 2.5.

2.1 Query Log

The query log of a keyword search engine is typically modeled as a set of records (q_i, u_i, t_i, C_i), where q_i is a query submitted by user u_i at time t_i, and C_i is the set of clicked URLs by u_i after q_i and before the user issues another query.

Following common practice [6,7,18], we can partition a query log into task-oriented *sessions*, where each session is a contiguous sequence of query records from the same user. Two contiguous queries are put in the same session if their time difference is at most t_θ (typically, t_θ is 30 minutes). Within the same session, we can assume that the user's search intent remains unchanged.

2.2 Obtaining Locations from a Query Log

Location Distribution of URLs. A webpage, corresponding to a URL, may contain information about one or more spatial locations. The *location distribution* of a URL d is a probability distribution p_d over a set of locations $L = \{(lat, lon) \mid lat, lon \in R\}$, where $\sum_{l \in L} p_d(l) = 1$. For the purposes of this paper, for each URL in the query log, we fetch the document and parse the content using GeoDict[1], a simple library/tool for pulling location information from unstructured text. This provides us with the location distribution for each URL. Alternatively, other methods for extracting locations from text can be applied [10].

Location Distribution of Queries. We also model the location distribution of a query q_i as a probability distribution p_{q_i}, and we can obtain it from a linear combination of the distributions of the clicked URLs for q_i. Formally,

$$p_{q_i}(l) = \frac{\sum_{d_j \in C_{q_i}} p_{d_j}(l)}{\sum_{l' \in L} \sum_{d_j \in C_{q_i}} p_{d_j}(l')},$$

where C_{q_i} is the set of clicked URLs for query q_i. In this paper, we use the location distributions of the queries to facilitate the problem of recommending queries to a search engine user u, that are not only semantically relevant to the query issued by u, but also are spatially close to the physical location of u.

[1] https://github.com/petewarden/geodict

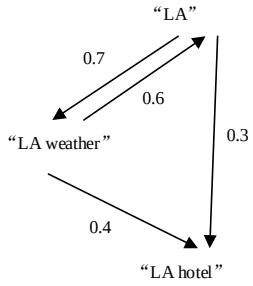


Fig. 1: QFG

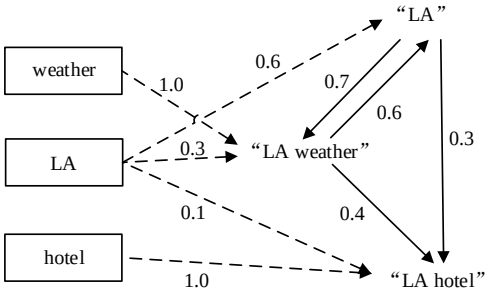


Fig. 2: TQGraph

2.3 Query-Flow Graph

One of the most promising directions for performing query recommendation relies on the extraction of behavioral patterns in query reformulation from query logs. The query-flow graph (QFG in short) [6] is a graph representation of query logs, capturing the "flow" between query units. Intuitively, a QFG is a directed graph of queries, in which an edge (q_i, q_j) with weight w indicates that query q_j follows query q_i in the same session of the query log with probability w.

More formally, QFG is defined as a directed graph $G_{qf} = (Q, E, W)$, where Q is the set of nodes, with each node representing a unique query in the log, $E \subseteq Q \times Q$ is the set of edges, and W is a weighting function assigning a weight $w(q_i, q_j)$ to each edge $(q_i, q_j) \in E$. In G_{qf}, two queries q_i and q_j are connected if and only if there exists a session in the query log where q_j follows q_i. Figure 1 illustrates a QFG with three query nodes.

Recommendation via QFG. Given a query $q \in Q$, the top-k recommendations for q can be obtained by a random walk with restart (RWR) [15] process starting from q, as suggested in [6]. At each step of the RWR, the random walker moves to an adjacent node with probability $1 - \alpha$ via the transition matrix W, or *teleports* to the original node q with probability α. In this way, a RWR process defines a Personalized PageRank score $PPR(q, q', W)$ for each node q' as the probability that the RWR starting from q reaches node q'. In this way, the top-k recommendations can be the set of k nodes q' in Q, which have the maximum PPR scores w.r.t. q in QFG. In other words, the recommendation score $rec_q(q')$ for each $q' \in Q$ is defined as:

$$rec_q(q') = PPR(q, q', W), \tag{1}$$

where W is the transition matrix for the PPR, and queries having the top-k scores are recommended.

However, QFG has an obvious disadvantage for query recommendation; that is, it cannot make any recommendation to an input query q, if $q \notin Q$. In

other words, if a query has not appeared in the query log before, QFG fails to generate any recommendation.

2.4 Term-Query-Flow Graph

Another popular method for query recommendation is the term-query-flow graph (TQGraph) [7], which basically extends the QFG method by considering a center-piece subgraph induced by terms contained into queries.

Formally, a TQGraph is a directed graph $G_{tqg} = G_{qf} \cup G_{tq}$, where G_{qf} is the QFG as described in Section 2.3, and G_{tq} is a bipartite graph of *term* nodes and *query* nodes. Specifically, the set of nodes in the TQGraph is $V_{tq} = Q \cup T$, where Q is the set of queries and T is the set of terms. Edge (t, q) exists in E_{tq}, if the term t is contained in query q. Figure 2 illustrates a TQGraph with three query nodes and three term nodes.

Recommendation via TQGraph. Given a query $q \in Q$, the top-k TQGraph recommendations for q are obtained by ranking all $q' \in Q$ based on their aggregate PPR scores w.r.t. each term $t \in q$. In other words, the recommendation score $rec_q(q')$ for each $q' \in Q$ is defined as follows:

$$rec_q(q') = \prod_{t \in q} PPR(t, q', W \cup E_{tq}) \tag{2}$$

We can see that TQGraph can generate recommendations for query q, as long as all the terms within q appear in the query log. Empirically, TQGraph has a much better *query coverage* compared to QFG, because it can also be used for queries that are asked for the first time.

2.5 Query Auto-Completion using a Prefix-Tree

ab	4
b	2
bba	1
caca	3
caccc	1
cbac	2
cbba	1

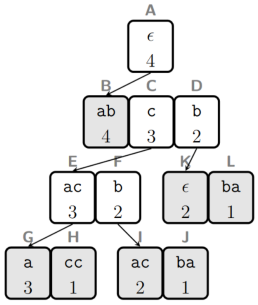


Fig. 3: An example of prefix-tree.

Query auto-completion is another important feature of search engines, which aims at providing instant useful query suggestion while the user is typing his/her query. Due to the nature of query auto-completion, the suggested query q' should have the input query q as its prefix. For example, if the input is $q =$ "*new yo*", the suggestion can be $q' =$ "*new york*" as "*new yo*" is a prefix of "*new york*".

To rank the feasible suggested queries, existing work [3] proposes using *most popular completion*. Specifically, given a prefix query q, let $C(q)$ denote the set of feasible suggested queries having q as a prefix. Then, MPC applies the following maximum likelihood estimation for ranking:

$$MPC(q) = \underset{q' \in C(q)}{\operatorname{argmax}} \frac{f(q')}{\sum_{q'' \in C(q)} f(q'')}, \tag{3}$$

where $f(q')$ is the frequency of q' in the query log.

To support efficient online retrieval of the most popular completion, a prefix-tree [13, 16] is used to organize the queries in a hierarchical structure. Formally, given a query log $\mathcal{L}$, a prefix-tree $\mathcal{T}$ is a tree, with each internal node v_{it} corresponding to a common prefix of queries, and each leaf node v_{lf} denoting a query $q \in \mathcal{L}$.

Figure 3 illustrates an exemplary prefix-tree. We can see that each leaf node (shaded) corresponds to a query in the log, and each internal node corresponds to the common prefix of all queries in its sub-tree. Note that ϵ here denotes the empty string.

To retrieve top-k completions according to Equation 3, we can perform an A* search on the prefix-tree. The basic idea is to derive an upper bound of the MPC score for each internal node by storing in it the maximal frequency of all leaves in its sub-tree. The reader can refer to [16] for more details.

3 Location-aware Query Recommendation

In this section, we present our location-aware query recommendation approach. We first introduce how we model the spatial proximity between a query and a user in Section 3.1. Then we introduce our query recommendation models in Section 3.2.

3.1 Spatial Proximity

Our recommendation goal is to provide query alternatives that are spatially close to the query issuer. For this, we should first define spatial proximity between a user located at l_u and a query with location distribution p_q. Some of the alternatives are:

- (i) Expected Distance (ED). Formally, $ED(p_q, l_u) = \sum_{l \in L(q)} p_q(l) \times dist(l, l_u)$.
- (ii) Min Distance (MinD). Formally, $MinD(p_q, l_u) = min_{l \in L(q)} dist(l, l_u)$.

– (iii) Max Distance (MaxD). Formally, $MinD(p_q, l_u) = max_{l \in L(q)} dist(l, l_u)$.
– (iv) Mean Distance (MeanD). Formally, $MeanD(p_q, l_u) = dist(mean(p_q), l_u)$.

However, these four distance-based measures are not fully consistent with our goal of finding spatially close queries to the user. For example, suppose we have two queries q_1 and q_2 with the following location distribution:

q_1 Hong Kong : 0.6, New York : 0.3, Los Angeles : 0.1
q_2 Beijing: 0.8, Los Angeles : 0.2

One would argue that q_1 is more spatially relevant to a user u_1 located in Hong Kong, compared to q_2, as a great portion of q_1's distribution is very close to u_1, which means that the majority of URLs related to query q_1 contain information that is spatially relevant to u_1. However, using ED, MaxD or MeanD might not select q_1, because after considering all locations of q_1, its overall distance to Hong Kong is quite large. At the same time, MinD does not distinguish q_1 and q_2 for a user u_2 located in Los Angeles, because MinD neglects the support probability of each location. Hence, we should define a more appropriate spatial proximity measure that better captures the distance between a query and a search engine user. In this direction, we propose the following measure:

Definition 1 Spatial Proximity sim_s. Given the location $l_u \in L$ of a user u, a range threshold r, and a location distribution p_q of a query q, the spatial proximity between l_u and q is the portion of p_q within distance r from l_u, i.e.,

$$sim_s(q, l_u) = \sum_{dist(l_u, l') < r} p_q(l')$$

The range threshold r models the distance that the user is willing to travel in order to use a service offered by the query results. In the example above, assuming that we select r as a within-city travel distance, we will have: $sim_s(q_1, l_{u_1}) = 0.6$, $sim_s(q_2, l_{u_1}) = 0$, $sim_s(q_1, l_{u_2}) = 0.1$, $sim_s(q_2, l_{u_2}) = 0.2$. This is consistent with the intuition that u_1 is more related to query q_1, and u_2 is more related to query q_2. In our experiments, we use $r = 100km$ by default and compare the performances of our methods for different values of r.

3.2 Location-aware Query Recommendation Models

After obtaining the spatial proximity between the user and the queries, we can adjust the weights on the edges of QFG to give higher preference to queries that have larger sim_s to the user.

Definition 2 Spatially Adjusted Weights. Given a query $q \in Q$ issued at location l_u, the spatially adjusted weight for an edge of a QFG $(q_i, q_j) \in E$ is defined as:

$$\tilde{w}(q_i, q_j) = \beta \times w(q_i, q_j) + (1 - \beta) \times sim_s(q_j, l_u)),$$

where β is a parameter that controls the relative importance of spatial proximity and $w(q_i, q_j)$ is the original weight of the edge (q_i, q_j) in the QFG.

With the linear function in Definition 2, we obtain a location-aware transition matrix $\tilde{W}$ from the original matrix W. Then, we can perform location-aware query recommendation based on $\tilde{W}$. In other words, the recommendation processes for spatial QFG (SQFG) and spatial TQGraph (STQGraph) are the same as their location-agnostic counterparts, except that we use $\tilde{W}$ instead of W.

Recommendation via SQFG. Given a query $q \in Q$ issued at location l_u, the top-k SQFG recommendations for q can be obtained by a location-aware Personalized PageRank (PPR) w.r.t. node q, i.e.,

$$rec_q(q') = PPR(q, q', \tilde{W}), \tag{4}$$

where $rec_q(q')$ is the recommendation score for query q' and $\tilde{W}$ is the location-aware transition matrix for the PPR.

Recommendation via STQGraph. Given a query $q \in Q$ issued at location l_u, the top-k STQGraph recommendations for q can be obtained by ranking all $q' \in Q$ based on their aggregate PPR scores w.r.t. each term $t \in q$. In other words, the recommendation score $rec_q(q')$ for each $q' \in Q$ is defined as follows:

$$rec_q(q') = \prod_{t \in q} PPR(t, q', \tilde{W} \cup E_{tq}) \tag{5}$$

3.3 BCA with Online Transition Matrix $\tilde{W}$

We extend the popular Bookmark Coloring Algorithm (BCA) [5] to compute the top-k PPR results based on the location-aware transition matrix $\tilde{W}$ on query time as our basic method. The basic idea of BCA is to model the RWR process as a bookmark coloring process, in which some portion of the ink in a processed node is sent to its neighbors, while the remaining ink is retained at the node.

Specifically, starting from the query node q with 1.0 units of ink, BCA keeps α portion in q and distributes the remaining $1 - \alpha$ portion to q's neighbors in the graph using the weights of the outgoing edges to determine the percentage of ink sent to each neighbor. The process is repeated for each node that receives ink, until the residue ink to be redistributed becomes a very small percentage of the original 1.0 units. Different from traditional PPR computation using BCA, our transition matrix $\tilde{W}$ can only be obtained online by Definition 2, after we know the location of the query issuer l_u.

In our implementation, the spatially adjusted weights of each edge (q_i, q_j) are also computed online based on l_u, at the time when the query node q_i is distributing ink. This means that the computation of $\tilde{W}$ is done during BCA simulation. A node distributes ink only if the quantity of the ink exceeds a

Algorithm 1: BCA

Input: Transition matrix W, starting node q, user location l_u
Output: Apprximated PPR vector rec_q

```
1  PriorityQueue que ← ∅
2  Add q to que with q.ink ← 1.0
3  R ← ∅;
4  Cache ← ∅;
5  while que ≠ ∅ and que.top.ink ≥ ϵ do
6      Deheap the first entry top from que;
7      R[top] ← R[top] + top.ink × α;
8      for q′ ∈ top.neighbors do
9          if q′ ∈ Cache then
10             sim_s(q′, l_u) ← Cache[q′];
11         else
12             Compute sim_s(q′, l_u) using Definition 1;
13             Cache[q′] ← sim_s(q′, l_u);
14         Compute w̃(top, q′) using Definition 2;
15         q′.buf ← top.ink × (1 − α) × w̃(top, q′);
16         if q′.buf ≥ ϵ then
17             Add q′ to que with ink q′.buf;
18             q′.buf ← 0;
19 return R
```

threshold ϵ (typically, $\epsilon = 10^{-5}$). BCA terminates when there are no more nodes to distribute ink.

We adopt the following two optimizations in our BCA implementation.

- **Lazy Updating Mechanism.** In the original BCA, a node distributes its ink aggressively, i.e, each neighbor q' of node *top* will be pushed into the priority queue *que* after receiving some portion of *top.ink*. On the other hand, we only care about the nodes with ink greater than ϵ. Based on these two observations, a lazy updating mechanism can reduce the number of non-necessary pushing without changing the final results; the pushing a node q' into the priority queue *que* is delayed until the ink it receives is greater than ϵ. If the amount of received ink is less than ϵ, q' only accumulates it in in a buffer; as soon as the buffer's ink exceeds ϵ, q' is pushed into *que*.
- **Spatial Proximity Caching**. Every time when we need to distribute ink to a query node q', we need to compute the spatial proximity between q' and the location of the user l_u. However, the same query node may be processed multiple times in a single BCA call. In view of this, we cache the spatial proximities between the location of the user l_u and the query nodes that have been computed so far. By doing this, we only need to compute the spatial proximity for a query q' once during a BCA call.

Algorithm 1 details our implementation of BCA, including the two optimizations mentioned above. Priority query *que* maintains the nodes to be processed in descending order of their ink (Line 1). *que* initially contains only

one node with ink amount 1.0 (Line 2), i.e., the starting node of the PPR. The nodes that have some retained ink are kept in a dictionary R, which is initially empty (Line 3). Termination conditions are checked at each iteration (Line 5). Within each iteration, we first dequeue from *que* the node with the most ink to distribute (Line 6). Then we leave α portion to its result (Line 7), and distribute the rest to its neighbors with weights $\tilde{w}$ (Lines 8-18). We first check the spatial cache whether q' has been computed before (Line 9). If so, the spatial proximity between q' and l_u can be directly got from the cache (Line 10). Otherwise we need to compute $sim_s(q', l_u)$ (Line 12) and save to the cache (Line 13). Finally, for each of the neighbors, if the received ink is greater than a threshold ϵ (Line 16), the corresponding query node will be pushed into *que* (Line 17) and the corresponding buffer is cleared (Line 18). Finally, the dictionary R is returned. In SQFG, where a single RWR search is applied, the k query nodes in R with the most retained ink are recommended. In STQGraph, the Rs of the RWR searches from all terms are summed up at each query node before computing and returning the top-k query nodes.

4 Location-aware Query Auto-Completion

In this section, we discuss how location-aware query auto-completion can be performed, based on the spatial proximity measure defined in Equation 1.

Recall that location-agnostic auto-completion methods essentially rank the candidate queries by descending order of the conditional probability:

$$sim_p(q', q) = \frac{f(q')}{\sum_{q'' \in Pre(q)} f(q'')}, \tag{6}$$

where $Pre(p)$ is the set of candidate completion queries that have q as their prefix.

For example, given the query log and prefix-tree in Figure 3, if the input query q = "c", the traditional location-agnostic method would suggest the following top-2 completions: q_1 = "caca" and q_2 = "cbac", because they have the largest sim_p to q.

Location-aware Auto-Completion. To integrate our spatial proximity into the auto-completion model, we consider a linear combination of sim_p and sim_s as follows:

$$sim_c(q', q) = \gamma \times sim_p(q', q) + (1 - \gamma) \times sim_s(q', l_u), \tag{7}$$

where γ is a parameter that controls the relative importance of spatial proximity.

4.1 Query Auto-Completion Algorithm

In this section, we first propose an upper-bound of sim_c in Equation 7. Then we present an A* search algorithm to support efficient retrieval of top-k query auto-completion based on the upper-bound.

Upper bound of $\mathbf{sim_c}$. To derive an efficient pruning algorithm to retrieve top-k completion, we need an upper-bound estimation of the ranking score sim_c in Equation 7. For an internal node v of the prefix-tree, we define its *exaggerated location distribution* as:

$$\hat{p}_v(l) = \max_{u \in L(v)} p_u(l)$$

, where $L(v)$ is the set of leaf nodes under v. Then we can define an upper bound for an internal node as follows:

Lemma 1 *Given an internal node v of prefix-tree T and the user's location l_u, the maximal sim_c score of all leaf nodes under v is not larger than*

$$UB(v) = \gamma \cdot v.\max(sim_p) + (1.0 - \gamma) \cdot \hat{sim}_s(v, l_u),$$

where $v.\max(sim_p)$ is the maximum sim_p score of leaf nodes under node v, and $\hat{sim}_s(v, l_u)$ is the exaggerated spatial proximity using $\hat{p}_v$.

Proof $UB(v)$ assumes the maximum possible sim_p with maximum sim_s, resulting in an upper bound of sim_c naturally.

Note that $UB(v)$ is not sensitive to the location of the user l_u, so it can be computed offline after we build the prefix-tree. During the online phase, we can direct access the upper-bound value $UB(v)$ of any internal node v without any additional computational cost.

Lemma 2 *Assume that $v_1\ v_2\ \cdots\ v_l$ is a path from the root v_1 to a leaf node v_l of the prefix-tree T. We have:*

$$UB(v_i) \geq UB(v_j),\ \forall i < j.$$

Proof By definition, $UB(v)$ derives the bound with two components, i.e., maximum sim_p and maximal sim_s, both of which come from its leaf nodes. Hence, we have $UB(v) = \max_{v' \in child(v)} UB(v')$, which means $UB(v_i) < UB(v_j)$ for $i < j$ is not possible.

AC-Pruning Algorithm. Based on Lemmas 1 and 2, we can design an A* algorithm for efficient top-k completion. The basic idea of AC-Pruning is to use $UB(v)$ value as a heuristic for pruning non-promising sub-trees.

Algorithm 2 shows the details of the algorithm. We first initialize the result set Q and a priority queue *que* (Line 1-2). Then we locate the internal node v for prefix q (Line 3). We push v into the priority queue with $UB(v)$ as the priority. We maintain a variable b as the k-th largest sim_c score so far, and initialize it as 0.0 (Line 5). During each iteration, we first get the top node in *que* (Line 7). If *top* is a leaf node, we compute its sim_c score and update the result set Q and b if needed (Lines 8-11). Else, for each child node of *top*, we push it into *que* with its UB value as priority key (Lines 12-14). We repeat this process until either *que* is empty or the largest priority value is no more than b.

Note that we can use another priority queue to efficiently maintain the result set Q and b at the same time.

Algorithm 2: AC-Pruning

```
Input: Query q, user location l_u, prefix-tree T
Output: Completion queries Q
PriorityQueue que ← ∅
Q ← ∅
Find v ∈ T for prefix query q
Add v to que with priority key UB(v)
b ← 0.0
while que ≠ ∅ and que.top.priority ≥ b do
    top ← pop(que);
    if top is leaf node then
        s ← sim_c(top, q);
        if s ≥ b then
            Update Q and b;
    else
        for q' ∈ top.children do
            Add q' to que with priority UB(q')
return Q
```

5 Approximate Spatial Proximity

Both our query recommendation and auto-completion methods require frequent computation of the spatial proximity sim_s (see Definition 1). To compute sim_s, we need to enumerate all locations of the query q' in order to accumulate the distribution. To reduce this high cost, we propose to compute a partitioning based approximation of $sim_s(q, l)$ as follows:

$$\hat{sim}_s(q, l) = \sum_{cir(l_u) \text{ intersects } c} p_q(c), \tag{8}$$

where c is a spatial partition of locations, $cir(l_u)$ is the circle with l_u as center and r as radius, and $p_q(c) = \sum_{l' \in c} p_q(l')$ is the location distribution of q that falls into partition c.

We use a grid to partition the space. Hence, locations that fall into the same cell belong to the same partition. If the length of each grid cell is a, to compute $\hat{sim}_s$, we only need to accumulate $p_q(c)$ for at most $\lceil \frac{2r}{a} + 1 \rceil^2$ partitions. In our experiments, we use $a = r$, so the computational cost is much lower than computing the exact sim_s, which requires enumeration of all locations.

Figure 4 illustrates an example of our partitioning based approximation. The dots with number next to them represent the location distribution of a query q. Suppose a user is located at the starred location and the circle is defined by that location and the range threshold r. The shaded cells are those which intersect the circle and according to Definition 1, $sim_s(q, l_u) = 0.2+0.05+0.3 = 0.55$. After using our spatial partition, we can obtain an approximation of

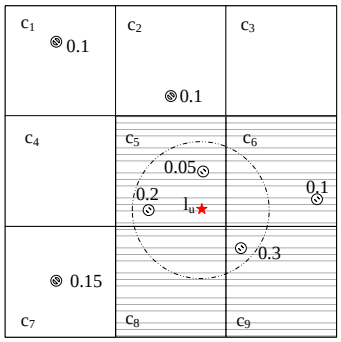


Fig. 4: Example of partitioning

Table 1: Location distribution approximation

cell	$p_q(c)$	cell	$p_q(c)$	cell	$p_q(c)$
c_1	0.1	c_2	0.1	c_3	0
c_4	0	c_5	0.25	c_6	0.1
c_7	0.15	c_8	0	c_9	0.3

the location distribution as in Table 1. Then, an approximation of the spatial proximity is computed as $\hat{sim}_s(q, l_u) = 0.25 + 0.1 + 0 + 0.3 = 0.65$.[2]

6 Experimental Evaluation

This section reports the experimental results of our study. Section 6.1 and Section 6.2 introduce our datasets and experimental methodology. Section 6.3 to 6.5 report the evaluation of query recommendation, while Section 6.6 reports the evaluation of query auto-completion.

6.1 Dataset

We use AOL in all our experiments. AOL is a well-known public query log from a major commercial search engine, which consists of Web search queries collected from 657k users over a two months period in year 2006. This dataset is sorted by anonymous user ID and sequentially arranged, containing 20M query instances corresponding to around 9M distinct queries. After we sessionize the query log with θ_t = 30min, we obtain a total of 12M sessions.

6.2 Methodology

Query Recommendation. We adopt the automatic evaluation process described in [18], to assess the performance of the tested methods. In a nutshell, we use part of the query log as training data to generate recommendations for a kept-apart query log fragment (the test data). In the test query log, we

[2] We do not further refine to get an exact result by looking into the locations within the cells, because we believe that those locations near the range r from the user are still spatially relevant (see the location in cell c_6 of Figure 4).

denote by $q_{i,j}$ the j_{th} query in session s_i. We assume that all $\{q_{i,j} \mid j > 1\}$ are good recommendations for query $q_{i,1}$, in accordance to previous work [18].

Specifically, we use 90% of the query log for training, which contains $11M$ sessions and $8.4M$ distinct queries. We use the remaining 10% of the query log to generate testing queries. We first extract sessions with at least two queries, and randomly sample 10K queries as our testbed. We take the first query of each session as input and the queries that follow as the ground truth recommendations. Formally, the ground truth for input query $q_{i,1}$ is $\{q_{i,j} \mid j > 1\}$, where $q_{i,j}$ is the j_{th} query appearing in the i_{th} session. While the objective of this evaluation approach may not necessarily be aligned with what a good recommendation could be for particular instances, by being entirely unsupervised and applied on a large number of sessions, it is a strong indicator of the techniques' performance. Note that we randomly assign the location of the query issuer l_u, as the dataset does not contain the location information of about the users.

We use the following three metrics to evaluate the performance of each method:

- *coverage*. This is the percentage of input queries that can be served with at least one recommendation.
- $precision@k$. This is the percentage of recommended queries in the top-k lists that are in the ground truth as described previously. Formally, $precision@k = \frac{\#HIT}{k \cdot \#query}$, where $\#HIT$ is the total number of recommended queries that are part of the ground truth, and $\#query$ is the number of input queries.
- $sim_s@k$. This is the average spatial proximity (see Definition 1) between the recommended queries in the top-k lists to the location of the query issuer l_u.

Query Auto-Completion. We randomly sample a set of 5000 queries from our query log, and use them as query inputs to test the performance of our query auto-completion. There are 1.577 words within each query on average. By default, we set $\gamma = 0.95$ and $k = 10$ (Google uses $k \leq 10$).

6.2.1 Competitors

For query recommendation, we compare the following approaches:

- **LKS** [23]. LKS is the most recently proposed location-aware keyword suggestion approach. It first builds a bipartite graph of queries and URLs using the query log, and then performs location-aware random walk over the graph during online recommendation. We use the default settings of LKS, i.e., the restart probability α_{LKS} and the edge weight adjustment parameter β_{LKS} are both set to 0.5.
- **SQFG**. SQFG is our spatial QFG method as described in Section 3. By default, we set the spatial radius threshold $r = 100$km, the restart probability $\alpha = 0.5$, and the spatial adjustment factor $\beta = 0.5$.

- **STQGraph**. STQGraph is our proposed spatial TQGraph approach as described in Section 3. We use the same default settings as in SQFG.
- **STQGraph***. STQGraph* is our spatial partitioning based approximation approach. By default, we use 100km as the length of each grid cell, and all the other parameters are same as STQGraph.

For query auto-completion, we compare the following three methods:

- **Basic**. It is a straightforward solution, which enumerates all leave nodes under the input query, computes their sim_c scores and finally finds the top-k out of them.
- **AC-Pruning**. It is out proposed approach in Algorithm 2.
- **AC-Pruning***. It is an approximated version that uses $\hat{sim}_s$ in Section 5 for spatial proximity.

6.3 User Study

We first conducted a user study to compare our proposed spatial proximity sim_s in Definition 1 with the four alternatives mentioned in Section 3. We first used our STQGraph to generate top-1 recommendations for 100 random queries with different spatial proximity measures. Then, for each of the recommended queries, we showed its location distribution as well as the location of the query issuer l_u to the participants. They were asked to rate the spatial relevance between the recommended query and the query issuer, using one of the following rating levels: 0 for not related at all, 1 for somehow related and 2 for very related. The recommended queries were shuffled before given to the participants, so that they could not know which spatial proximity measure was used to generate the recommended query. We asked 15 participants (HKU students) to rate the recommended queries, and each of the queries were given at least 3 ratings.

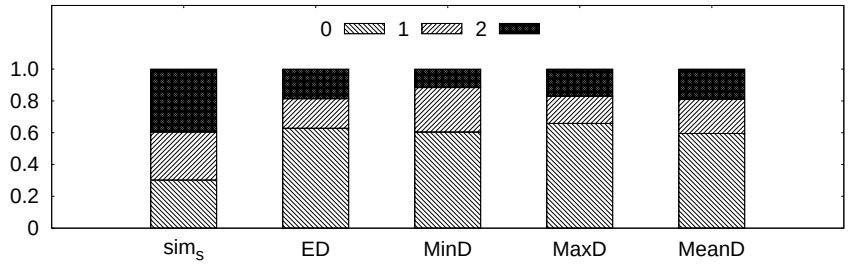
The results are shown in Figure 5. We can see that our sim_s has the largest percentage of 2s, which means that sim_s is acknowledged to be the best measure of spatial proximity. Out of the four alternatives, ED and MeanD received relatively better user feedback. This is because a smaller ED or MeanD implies smaller overall distance to the query issuer. MinD got the worst user feedback because MinD only considers the location l which is the closest to l_u, but not the probability $p_q(l)$, which could be too small.

From this user study, we can conclude that users prefer sim_s over the other four proximity measures. In the rest of our experiments, we use sim_s to evaluate the spatial quality of recommended queries.

6.4 Effectiveness

We first compare the four tested methods. Then we test the parameter sensitivity of our STQGraph method to different parameter values.

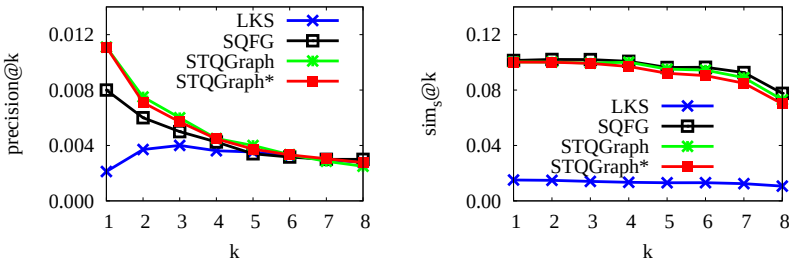
Fig. 5: User Study on Different Spatial Proximity Measures



6.4.1 Comparison Results.

Table 2: *coverage* results

method	LKS	SQFG	STQGraph	STQGraph_P
coverage	36.8%	27.9%	37.1%	37.1%



(a) *precision@k* (b) sim_s@*k*

Fig. 6: Comparison.

Table 2 compares all methods in terms of their *coverage*. We can see that SQFG has relatively low *coverage* compared the other three approaches. This is expected, because SQFG has the same disadvantage as QFG, i.e., it cannot provide recommendations to any previously unseen queries. STQGraph and LKS have similar *coverage*, much higher than that of SQFG. Note that STQGraph and STQGraph* have the same *coverage*, since our spatial partition based approximation only affects the ranking of the recommended queries.

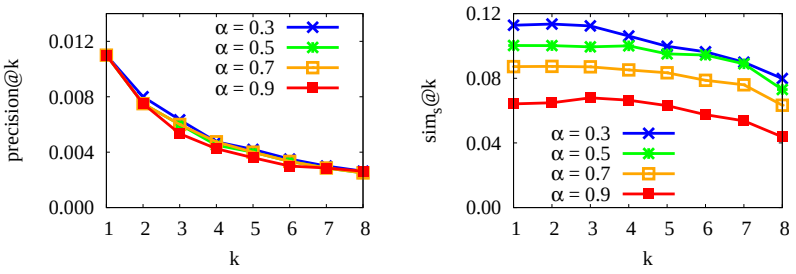
Figure 6(a) shows the *precision@k* of all methods. Since typical search engines (e.g. Google and Yahoo!) show eight recommendations, we test values of k from 1 to 8. Observe that our STQGraph and STQGraph* methods

have significantly larger *precision*@k compared to LKS. As k becomes larger, *precision*@k becomes smaller. This means our recommendation methods rank the recommended queries reasonably, as those with smaller ranks are more precise. STQGraph* has almost the same *precision*@k as STQGraph, which means that our spatial partition based approximation does not harm the recommendation quality in terms of semantic quality. The precision of SQFG is lower than that of STQGraph for small values of k.

Figure 6(b) shows the results of sim_s@k. Similar to the case of *precision*@k, sim_s@k drops as k increases. LKS has very poor sim_s@k result compared with our approaches. This is because LKS tends to recommend queries that share the same clicked URLs with the input query, without directly considering the location distribution of the recommended queries. SQFG, STQGraph* and STQGraph achieve almost the same sim_s@k.

6.4.2 Parameter Sensitivity.

We now test the sensitivity of our STQGraph approach to the values of its parameters. As *coverage* result is only related to the connectivity of STQGraph and is not sentitive to the parameters, we only show the *precision*@k and sim_s@k results.

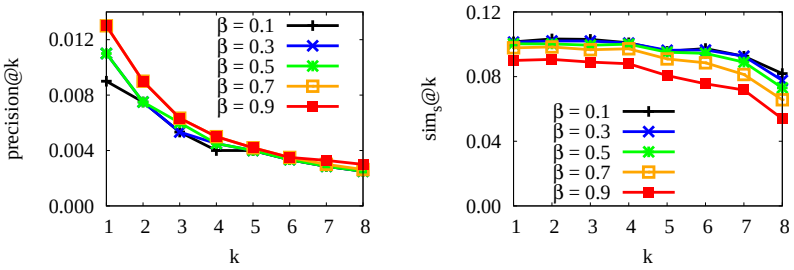


(a) *precision*@k (b) sim_s@k

Fig. 7: Varying α

• **Varying** α. Figure 7 shows the quality of STQGraph for various values of α. Observe that α does not influence *precision*@k very much. In addition, larger α leads to smaller sim_s@k. This is because a larger α gives higher weight to the adjacent queries to the input query in the graph, whereas potentially better queries exist at a larger distance.

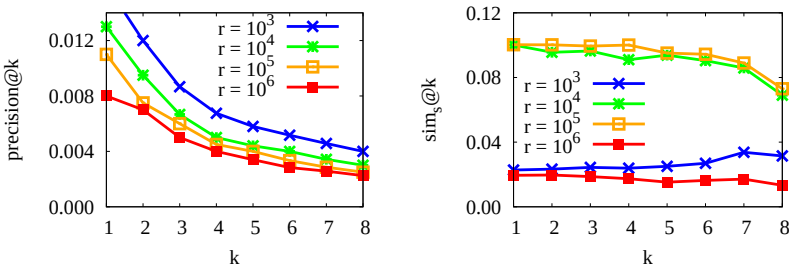
• **Varying** β. Figure 8 shows the quality of STQGraph for various values of β. We can see that larger β values lead to slightly higher *precision*@k. However, larger β values lead to smaller sim_s@k. This is because a larger β gives higher weight to the semantic relevance between queries and lower weight to the



(a) *precision*@*k* (b) sim_s@*k*

Fig. 8: Varying β

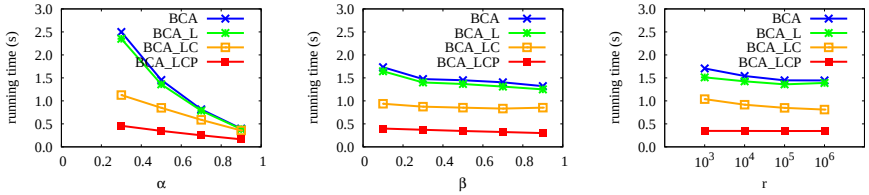
spatial proximity. Overall, a value of β close to 0.5 strikes a balance between the two factors giving good *precision*@*k* and sim_s@*k* at the same time.



(a) *precision*@*k* (b) sim_s@*k*

Fig. 9: Varying r

• **Varying** r. Figure 9 plots the quality of STQGraph for various values of r. A larger r leads to a smaller *precision*@*k*. This is because a larger r will result in larger spatial proximity in general, which eventually puts less emphasis to the original weights on the edges of QFG. From Figure 9(b), we observe that too large and too small r values lead to worse spatial proximity results. When we use a very small r, we get very small spatial proximity in general, leading to a worse sim_s@*k* result. When we use a very large r, we cannot distinguish queries that are actually close to the user, also leading to a worse sim_s@*k*. This result shows that we should choose an appropriate r for our method. Empirically, $r = 10^6$ (100 km) gives good results.



(a) varying α (b) varying β (c) varying r

Fig. 10: Running time of query recommmendation.

6.5 Efficiency of Query Recommendation

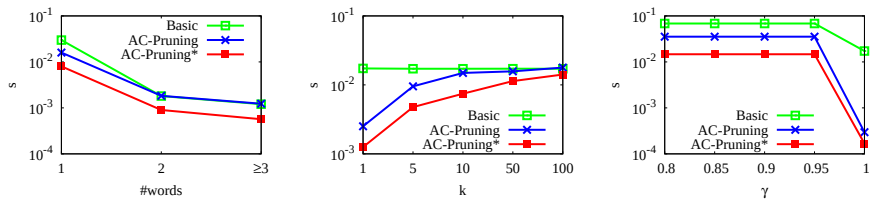
Now we test the efficiency of our optimizations and the approximation technique. We compare the overall running time of our STQGraph recommendation method, implemented with four versions of BCA for this purpose:

- **BCA**. The basic BCA algorithm without any optimizations.
- **BCA_L**. The BCA algorithm with the lazy update mechanism.
- **BCA_LC**. The BCA algorithm with the lazy update mechanism and spatial caching.
- **BCA_LCP**. The BCA algorithm with the lazy update mechanism, spatial caching and spatial partitioning based approximation. Note that this method corresponds to our STQGraph* method, which returns slightly different recommendations to STQGraph.

• **Varying** α. Figure 10(a) shows the average running time of STQGraph using the different versions of BCA for different values of α. We can see that all four versions terminate faster for larger values of α, which is consistent with our intuition. BCA_LCP is significantly faster than all other versions. When $\alpha = 0.5$, it takes only 0.3s for a query, which indicates that our STQGraph* can provide instant query recommendations.

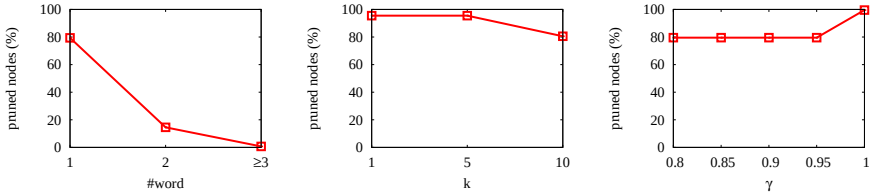
• **Varying** β. Figure 10(b) shows the running times for different values of β. A first observation is that the cost of the different versions of BCA is not much sensitive to β, as β only determines how much importance we put to spatial proximity. For the default values of α and r, BCA_LC takes around 1.0s for each query, while BCA_LCP needs only 0.3s. Considering that STQGraph* achieves similar effectiveness to STQGraph, as shown in our previous experiments, STQGraph* (which uses BCA_LCP) is more suitable for real-time applications.

• **Varying** r. Figure 10(c) shows the running times for different values of r. Observe that the runtimes for all methods are not sensitive to the change of r. This is because r only influences the values of spatial proximity sim_s.



(a) varying #words (b) varying k (c) varying γ

Fig. 11: Running time of query auto-completion.



(a) varying #words (b) varying k (c) varying γ

Fig. 12: Percentage of nodes being pruned.

Table 3: #candidates v.s. #words

#words	1	2	3
#candidates	362.8	49.0	29.7

6.6 Efficiency of Query Auto-Completion

Varying #words. Figure 11(a) reports the running time of the methods as a function of the number of words in the input query q. We can see that the cost is lower for longer queries. This is because a longer prefix query corresponds to a node which is deeper in the prefix-tree which has a smaller number of descendants. We can also see that our approximated method boosts the efficiency. Both AC-Pruning and AC-Pruning* terminate within 0.01s, being able to provide instantaneous reformulations.

When the number of words equals 1, our two approaches are significantly faster than Basic, but AC-Pruning has similar performance as Basic when there are more than one words. Figure 12(a) further shows the percentage of nodes being pruned v.s. the number of words in the input query. Note that about 80% of nodes are pruned when #words = 1, and few can be pruned when there are more than one words, which is consistent with our running time results. This is because there are not many candidate queries when #words > 1 (as shown in Table 3). Given that auto-completion is more useful when the input query is shorter and the number of potential candidates is large, our pruning technique can be considered effective.

Varying k. Figure 11(b) shows the average running time as a function of k on the whole set of 5000 input queries. As expected, the running time increases for larger values of k. This is because larger k values lead to lower sim_c scores, which delay the termination of the algorithm. When k is smaller, our pruning techniques gain larger improvement compared to Basic. This is because more nodes can be pruned with a smaller k, as shown in Figure 12(b).

Varying γ. Figure 11(c) shows the average runtime as a function of γ on the whole set of 5000 input queries. Observe that γ does not affect the cost too much, except when $\gamma = 1.0$. This happens because $\gamma = 1.0$ means that we do not need to compute the spatial proximity sim_s at all. Figure 12(c) shows the percentage of nodes being pruned. The results are consistent with Figure 11(c).

7 Related Work

Query recommendation and query auto-completion are two important add-on functions of search engines; they both aim at providing accurate query reformulation suggestions on-the-fly. In this paper, we focus on both problems, and provide solutions to address there location-aware settings.

For *query recommendation*, there have been many works, and most of them rely on analyzing query logs to extract useful patterns that model user behavior. All these works boil down to modeling the similarity between queries, often using random walk based proximity measures on graphs that may include users, terms, queries and URLs.

Early approaches rely on clustering similar queries [1, 26], where the similarity is defined using the query-URL graph or the term-vector representations of queries obtained from the clicked URLs. Later, [28] proposed the extraction and analysis of search sessions from the query log that capture the causalities between queries, and combined this with content-based similarity. In [2], the authors introduced the concept of *cover-graph*, a bipartite graph between queries and Web pages, where links indicate the corresponding clicks. [14] proposes recommending queries in a structured way for better satisfying exploratory interests of users, and [27] proposes a context-aware query recommendation model considering the relationship between queries and their clicks.

[6] and [7] are two of the most influential works in query recommendation. Both of them exploit flow patterns in query logs, and use graph-based methods to perform query recommendation. [6] builds a graph of queries, termed the query-flow graph (QFG), in which the links model the transition probabilities between queries. [7] further extends the QFG to a term-query-flow graph (TQGraph), which also include nodes representing terms within queries. In this way, TQGraph can provide recommendations even for queries that never appeared in the query log. In both works, the top-k recommendations are obtained by performing random walk with restart (RWR) in the graphs.

For *query auto-completion*, only the most relevant *expansions* of the input query are shown, typically while the user is typing the query. Most existing

works apply prefix-based recommendations and use trie-like index structures [4,8,24,25]. These papers either consider a different ranking function, or focus on improve the efficiency of the searching algorithm. [20] considers location-sensitive query completion, but it assumes that each query belongs to a single location, which is not practical in real application. [30] addresses a similar problem to spatial query auto-completion, while the output of its problem is spatial objects (e.g., POIs) instead of keyword queries. Recently, [17] studied an auto-completion problem, which assumes a set of spatial objects. On the other hand, we only have to consider the location information of queries.

Although many keyword search queries are sent from mobile users who have spatial search intent, there is limited research on location-aware query reformulation. In [22], the similarity between two queries is considered high if there are groups of similar users issuing these queries from nearby places at similar times. Google [21] keeps track of the locations where past queries are issued and determines the similarity between queries by also considering the proximity between the locations of the corresponding query issuers. [29] apply a learning model on the tensor representation of the user-location-query relations to predict the user's search intent. Recently, [23] proposes a location-aware keyword suggestion (LKS) method, which extends the idea of [11]. However, LKS only considers the location information for the documents (URLs), without considering that of queries. As we argue in this paper, it is more important to consider the spatial proximity between the user and the queries than the documents, because it is the queries we recommend to the user in the task of query recommendation. We experimentally compare our proposed methods with LKS and show that our methods provide better recommendations. A preliminary version of this paper [19] has addressed location-aware query recommendation, while this paper also considers location-aware query auto-completion.

8 Conclusion

We study the problem of location-aware query reformulation for search engines, including query recommendation and query auto-completion. We first propose a spatial proximity measure between a keyword search query and a search engine user. For *query recommendation*, based on this proximity measure, we extend two popular query recommendation approaches (i.g., QFG and TQGraph) to apply for the location-aware setting. In this way, we can generate recommendations that are not only semantically relevant, but also spatially close to the query issued by a user at a specific location. In addition, we extend the Bookmark Coloring Algorithm to support efficient online query recommendation. For *query auto-completion*, we formulate the problem by combining the original frequency-based ranking function and our location-based ranking function. We further propose an A* search algorithm to efficiently solve the problem. Furthermore, we propose approximate versions of the algorithms that use spatial partitioning to approximate and accelerate the

computation of our proposed spatial proximity measure. Experiments on a real query log show that our proposed methods significantly outperform previous work in terms of both semantic relevance and spatial proximity, and that our methods can be applied to provide reformulations within only a few hundreds of milliseconds.

Acknowledgments

We thank the reviewers for their valuable comments. This work is partially supported by GRF Grant 17205015 from Hong Kong Research Grant Council. It has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 657347.

References

1. R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology - EDBT Workshops*, 2004.
2. R. A. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, 2007.
3. Z. Bar-Yossef and N. Kraus. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web*, pages 107–116. ACM, 2011.
4. Z. Bar-Yossef and N. Kraus. Context-sensitive query auto-completion. In *WWW*, 2011.
5. P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Math*, 3:41–62, 2006.
6. P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *CIKM*, pages 609–618. ACM, 2008.
7. F. Bonchi, R. Perego, F. Silvestri, H. Vahabi, and R. Venturini. Efficient query recommendations in the long tail via center-piece subgraphs. In *SIGIR*, pages 345–354. ACM, 2012.
8. F. Cai, S. Liang, and M. de Rijke. Time-sensitive personalized query auto-completion. In *CIKM*, 2014.
9. 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*, pages 875–883, 2008.
10. Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pages 277–288, 2006.
11. N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR*, pages 239–246. ACM, 2007.
12. D. Downey, S. T. Dumais, and E. Horvitz. Heads and tails: studies of web search with common and rare queries. In *SIGIR*, 2007.
13. H. Duan and B.-J. P. Hsu. Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, pages 117–126. ACM, 2011.
14. J. Guo, X. Cheng, G. Xu, and H. Shen. A structured approach to query recommendation with social annotation data. In *CIKM*, pages 619–628. ACM, 2010.
15. T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526. ACM, 2002.
16. B.-J. P. Hsu and G. Ottaviano. Space-efficient data structures for top-k completion. In *Proceedings of the 22nd international conference on World Wide Web*, pages 583–594. ACM, 2013.
17. S. HU, C. XIAO, and Y. ISHIKAWA. An efficient algorithm for location-aware query autocompletion. *IEICE TRANSACTIONS on Information and Systems*, 101(1):181–192, 2018.

18. Z. Huang, B. Cautis, R. Cheng, and Y. Zheng. Kb-enabled query recommendation for long-tail queries. In *CIKM*, pages 2107–2112, 2016.
19. Z. Huang and N. Mamoulis. Location-aware query recommendation for search engines at scale. In *International Symposium on Spatial and Temporal Databases*, pages 203–220. Springer, 2017.
20. C. Lin, J. Wang, and J. Lu. Location-sensitive query auto-completion. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 819–820. International World Wide Web Conferences Steering Committee, 2017.
21. J. Myllymaki, D. Singleton, A. Cutter, M. Lewis, and S. Eblen. Location based query suggestion, Oct. 30 2012. US Patent 8,301,639.
22. X. Ni, J. Sun, and Z. Chen. Mobile query suggestions with time-location awareness, May 31 2012. US Patent App. 12/955,758.
23. S. Qi, D. Wu, and N. Mamoulis. Location aware keyword query suggestion based on document proximity. *TKDE*, 28(1):82–97, 2016.
24. M. Shokouhi. Learning to personalize query auto-completion. In *SIGIR*, 2013.
25. M. Shokouhi and K. Radinsky. Time-sensitive query auto-completion. In *SIGIR*, 2012.
26. J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *WWW*, 2001.
27. X. Yan, J. Guo, and X. Cheng. Context-aware query recommendation by learning high-order relation in query logs. In *CIKM*, pages 2073–2076. ACM, 2011.
28. Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW*, pages 1039–1040, 2006.
29. Z. Zhao, R. Song, X. Xie, X. He, and Y. Zhuang. Mobile query recommendation via tensor function learning. In *IJCAI*, pages 4084–4090, 2015.
30. Y. Zheng, Z. Bao, L. Shou, and A. K. Tung. Inspire: A framework for incremental spatial prefix query relaxation. *IEEE Transactions on Knowledge and data Engineering*, 27(7):1949–1963, 2015.