Joint Search by Social and Spatial Proximity

Kyriakos Mouratidis, Jing Li, Yu Tang, and Nikos Mamoulis

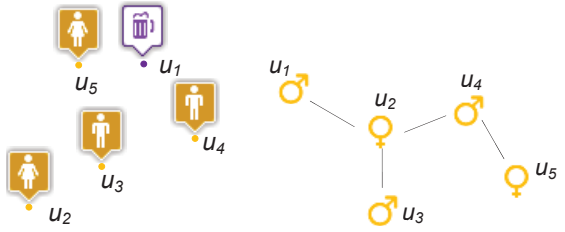
Abstract—The diffusion of social networks introduces new challenges and opportunities for advanced services, especially so with their ongoing addition of location-based features. We show how applications like company and friend recommendation could significantly benefit from incorporating social and spatial proximity, and study a query type that captures these two-fold semantics. We develop highly scalable algorithms for its processing, and enhance them with elaborate optimizations. Finally, we use real social network data to empirically verify the efficiency and efficacy of our solutions.

1 Introduction

The emergence of social networks (*SN*s) brings a new era in the organization and browsing of online information. Manufacturers and service providers are becoming increasingly interested in exploiting popular *SN*s to promote their products and services. Recently, Microsoft's search engine (Bing) has integrated social information from Facebook to return web pages that are popular among the friends of users [1]. Studies like [2] have investigated the influence between users of *SN*s and quantified the probability of a user performing an action (e.g., purchase a product) after his/her friend(s) did. Current text search systems have also incorporated social influence into query processing by taking into account friend relationships for the ranking of documents/objects [3], [4].

On the other hand, location-based services are an indispensable feature in *SN*s. This fact becomes increasingly prominent as the number of users who access *SN* applications on mobile devices is growing steadily. The most popular *SN*, Facebook, includes a set of location-based features, while others (such as Foursquare) are explicitly based on the management of user locations. Motivated by this trend, we investigate the integration of social and spatial information in a single query.

Consider a service like badoo.com, where a user u_1 who is looking for company to have lunch or watch a movie, may browse the profiles of nearby users and invite them to join him/her. Existing systems apply a traditional k-nearest neighbor query [5], potentially with some binary conditions (regarding age, sex, etc), to provide u_1 with the profiles of users in the vicinity. While recommended users are indeed near u_1 geographically, his/her true preferences of companions would be better captured if *SN* information was also taken into account. Assume, for example, that the users' Euclidean coordinates and social connections are as shown in Figures 1(a) and 1(b) respectively. The closest user to u_1 in the spatial domain is u_5. However, u_4 might be a better match because he locates only slightly farther (compared to u_5) but is "closer" in the social network. Conversely, the closest user socially (u_2) may be too far spatially. Therefore, to provide meaningful recommendations, both *social proximity* and *spatial proximity* should be incorporated into the search.



(a) Spatial domain (b) Social network

Fig. 1. Motivation example

In this paper we propose and study the *social and spatial ranking query* (*SSRQ*). *SSRQ* reports the top-k users in the *SN* based on a ranking function that incorporates social and spatial distance from the query user. Our key contributions are:

- We conduct the first study on a joint search by social and spatial user proximity.
- We propose a suite of processing methodologies, including a highly scalable and robust approach that relies on indexing and social summaries.
- We equip the latter with sophisticated optimizations, based on computation sharing, intermediate result caching and an accuracy-enhancing strategy that complements social summaries in proximity estimation.
- We use real *SN* data to experimentally evaluate our algorithms.

- *K. Mouratidis is with the School of Information Systems, Singapore Management University.*
  E-mail: kyriakos@smu.edu.sg
- *J. Li, Y. Tang, and N. Mamoulis are with the Department of Computer Science, University of Hong Kong.*
  E-mail: jli@cs.hku.hk, ytang@cs.hku.hk, nikos@cs.hku.hk

2 Related Work

2.1 Social Influence and Proximity Measures

The influence between two users captures the probability that one user follows the other's actions. The influence information stemming from *SN*s can improve marketing strategies, for instance, by recommending products to users based on the purchases of their contacts [6]. Existing work focuses primarily on finding the top-k most influential users from a graph of influence scores [2] or learning the influence scores based on users' past propagation of actions [7], [8]. Recently, [9] proposed an approach to directly obtain the top-k most influential users from historical data, without the intermediate step of constructing an influence graph.

Many measures are proposed for computing the influence between two users (vertices) in a social graph. Simple measures rely either on the shortest path distance or on vertex neighborhoods – i.e., the social proximity of two users may be defined as the inverse of their shortest path distance in the *SN* [3], [4] or as the number of their common friends [10]. Sophisticated proximity measures involve a combination of infinite sums over the ensemble of all paths between two vertices and their common neighbors (e.g., Katz measure, rooted PageRank, escape probability); [11] is an extensive survey on this subject.

2.2 Multiple-domain Search

Objects associated with multiple domain attributes have attracted considerable research interest. Web pages with geographic information and Flickr photos with geo-tags require query processing on both the spatial and textual domains. *Spatial keyword search* [12] retrieves objects that are not only close to users in physical space but which also match a set of keywords. [13] proposed the *IR*-tree data structure, which extends the R-tree with inverted files. This index can be used to efficiently support novel types of spatio-textual queries (e.g., [14]). A similar data structure appears in [15] to support a reverse form of spatial keyword search.

Location-based *SN*s, such as Foursquare, Whrrl and Gowalla, record users' location history (e.g., check-ins). In [16], Scellato et al. observe that in a location-based social network, 30% of new friends made are *place friends*, i.e., individuals who have visited the same places. Hence, they build a supervised learning framework which predicts new friend links based on the number of common contacts and common check-ins. In [17], Ye et al. observe that if a place is visited by friends of a user, this place is probably of higher interest to the user. Also, if a user has many nearby check-ins with another, one's preferred places may also be appealing to the other. These two factors, combined with potential geographic influence among the places themselves, are used to make location recommendations to *SN* users. The result in [17] is a set of places of interest, while in our problem the result comprises k other *SN* users.

Both aforementioned studies measure social proximity between users as the number of their common friends. In our case, social influence is extended to more than two hops in the *SN* and defined according to shortest path distance in the social graph. Also, in the spatial domain, both [17] and [16] consider historical check-ins, whereas we only consider the current locations of users, targeting present-time applications.

Armenatzoglou et al. [18] propose a general framework for queries over geo-social network data. They decouple the storage of the social network data from the geographic data; queries are evaluated as in a distributed database. The system supports searching for crisp structural patterns that appear in the social graph, which are spatially ranked. For example, the "nearest friends" query finds the k friends of a user u who are closest to a given location q. Our problem and solutions are different, since we integrate ranking at both social and spatial dimensions.

In [19], Cho et al. analyze the location data of *SN* users and notice that an individual's periodic movements which may seem random, are actually likely to correlate with the movements of his/her social contacts. This leads to a model of human mobility based on social links. Another related application is *proximity detection* in *SN*s. The goal is to continuously report to each user who, among his/her friends, are within a certain distance from the user's current location. The problem was introduced in [20] in the context of a P2P network. Subsequent approaches include dead reckoning [21] and adaptive safe region techniques [22], as well as constraint detection formulations [23]. Proximity detection considers only immediate friends of users and a fixed radius around their locations. In contrast, in *SSRQ* the result may include users at an unpredictable number of social hops and at variable spatial distances from the query user.

Bao et al. [24] propose a location-aware news-feed system. This enables users to browse spatially related messages from their friends or registered news sources. Unlike *SSRQ* (which selects users), that system filters news-feeds/messages. Also, it considers only immediate (1-hop) friends and news sources.

2.3 Shortest Path and Distance Computation

A traditional type of graph search is shortest path computation from a source to a target vertex. Dijkstra's algorithm starts from the source and iteratively expands the network using a priority heap, until the target is reached. To prune the search space and direct the graph expansion, A^* algorithm prioritizes the visiting order of nodes by estimating their distance to the target. [25] introduces the *landmark* approach which selects a set of vertices as landmarks in the

graph and pre-computes distances from every vertex to each landmark. Given two vertices and their distances to a specific landmark, the triangular inequality produces a lower bound on the distance between the two vertices. Using multiple landmarks, we derive an equal number of lower bounds, among which the tightest can be used to enhance A^* search. [26] extends this approach using a hierarchy of landmarks.

An approach to compute approximate distances between vertices in a graph is to construct oracles which provide constant query time while having linear space requirements. Theoretical results on distance oracles appear in [27], [28]. Sarma et al. [29] propose landmark based oracles which guarantee the theoretical result of [28] and experimentally outperform [27] and [28]. Distance oracles are not effective in our problem, which involves distance computations in a social graph, because we require exact distances, not approximate. Moreover, the theoretical error bounds of distance oracles are too loose and they are known to be poorly suited for social networks [29]. In [30], Cheng et al. propose a 2-hop cover data structure which supports efficient distance queries for a general graph with $O(|V||E|^{1/2})$ space. It is inapplicable to our setting because, for the density and scale of real *SN*s, its space requirements are prohibitive.

2.4 Top-k Processing

Our problem is related to top-k processing. A top-k query specifies a preference function f over the m attributes of a dataset and retrieves the k tuples that minimize (or maximize) this function. A thorough survey of top-k processing techniques is given in [31]. Here we survey the *threshold algorithm* (TA) and its variants [32] due to their higher relevance to *SSRQ*.

Assume that there are m repositories (sorted lists), one for each of the data attributes. The repository for the i-th attribute keeps all tuple identifiers sorted in ascending order of the i-th attribute. Two types of access are possible on each repository, *sorted* and *random*. Sorted access allows serial retrieval of elements (i.e., pairs of tuple identifier and its i-th attribute value) by iterative "get-next" operations, starting from the first element in the list, then moving to the second, etc. On the other hand, random access allows retrieving the attribute of any tuple in a repository directly.

TA requires that the preference function f is increasingly monotone on all m attributes. It probes (via sorted access) the repositories in a round-robin fashion. For each element pulled from a list, it computes the f value of the corresponding tuple by fetching its remaining $m-1$ attributes from the other repositories via random accesses. It maintains an interim result of the top-k tuples seen so far. It also keeps a threshold τ computed as the value of f over the last attribute values pulled from each of the m repositories. Essentially, τ is a lower bound on the f value of any non-encountered tuple further down the lists. TA terminates when τ is no smaller than any of the f values in the interim result (which is then reported as the final result).

TA assumes that random access is possible. NRA is the *no random access* version of the algorithm, where only sorted access is available on the repositories. The repositories are probed in round-robin order. For every encountered tuple, NRA maintains a lower and an upper bound of its f value. The lower bound (the upper bound) is computed by replacing the unseen attributes of the tuple with the last value pulled from the corresponding repository (the maximum possible value in the corresponding repository). NRA terminates when the k smallest upper bounds among seen tuples are no greater than the lower bound of any other encountered tuple.

Another variant of TA is the *combined algorithm* (CA). TA assumes that random and sorted access have the same cost. CA, instead, considers that random access is costlier than sorted. It proceeds similarly to NRA, but it periodically performs one random access. Specifically, for every κ sorted accesses, one random is made; κ is set to the ratio of random access cost to sorted access cost.

Bruno et al. [33] consider top-k queries in web-accessible databases. The data consists of a sorted list and a set of random access lists. Since random access is expensive, the authors propose that when an object is encountered in the sorted list, only a selected subset of the random access lists is probed to refine the object's f value bounds.

3 Problem Setting

The problem setting includes a set of users U and an undirected social graph $G = (V, E)$. Each user $u_i \in U$ has spatial coordinates in Euclidean space. The users may move dynamically; our system/query only considers their current (i.e., last reported) location. The social graph G includes a vertex $v_i \in V$ for every user $u_i \in U$. We establish the convention that vertex v_i corresponds to user u_i, i.e., the mapping is implied by the subscripts. We do not unify the two notations to help distinguish between spatial and social context. Every edge (v_i, v_j) in E represents a friend relationship between users u_i, u_j and is associated with a numerical weight that indicates the strength of the relationship – the smaller the weight, the stronger the friendship. In previous work, given the topology of a social network, the weights are mined from past propagation of user actions [7], [8]. We make no assumption about the weights other than them being positive numbers. We consider that G is undirected, but our work extends to directed graphs easily.

3.1 Ranking Function

We define *spatial proximity* between users u_i and u_j as their Euclidean distance $d(u_i, u_j)$. On the other hand,

TABLE 1
Frequently Used Notation

Notation	Explanation
$G(V, E)$	graph G with vertex set V and edge set E
$d(u_i, u_j)$	Euclidean distance between u_i and u_j
$p(v_i, v_j)$	graph distance between v_i and v_j
α	preference param. for social/spatial proximity
k	number of users to be reported by the query
R	the result set of the query
f_k	the k-th (i.e., maximum) f value in R
M	number of landmarks used
m_{ij}	graph distance between v_i and j-th landmark
s	partitioning granularity in grid index of *AIS*

we measure *social proximity* between vertices v_i and v_j based on their shortest path distance in G, and denote it as $p(v_i, v_j)$. We use this formulation because (i) it is simple and (ii) it is demonstrated to effectively capture social proximity/influence [3], [4].

Following common practice in combining measurements from different domains, we apply a linear function over the (normalized) social and spatial proximity to rank users [13], [34], [14]. Specifically, given a query user u_q, the ranking of $u_i \in U$ is determined by function f as:

$$f(u_q, u_i) = \alpha \cdot p(v_q, v_i) + (1 - \alpha) \cdot d(u_q, u_i) \quad (1)$$

where α is a (user- or application-specified) real number between 0 and 1 that determines the relative significance of proximity in the two domains. The smaller the value of f for a user, the more suitable he/she is for u_q. Note that our definition (and implementation) uses normalized social and spatial proximities, by dividing $d(u_q, u_i)$ and $p(v_q, v_i)$ with the maximum pairwise distance in Euclidean space and in the social graph respectively. For simplicity, we omit the denominators from the presentation.

3.2 Query Formulation

In this work, we propose the *social and spatial ranking query* (*SSRQ*) where a user u_q (or an application) provides parameter α and asks for the top-k users who minimize function f, with respect to his/her current location and social links. Formally:

Definition 1 (SSRQ): Given a set of users U, the underlying social graph G, a query user $u_q \in U$, and a preference parameter α, *SSRQ* returns the k users u in $U - \{u_q\}$ with the smallest $f(u_q, u)$ values.

That is, for every $u' \notin R$ and $u' \neq u_q$ it holds that

$$f(u_q, u') \geq f_k$$

where f_k is the maximum (i.e., least preferable) ranking value f across all users in the result R of the query. In Table 1 we summarize the frequently used notation.

4 Preliminary Solutions

We first present two simple solutions, namely *Social First Approach* and *Spatial First Approach*; then we hybridize them into an elaborate solution called *Twofold Search Approach*.

4.1 One Domain Approach

Social First Approach. A preliminary approach for *SSRQ* processing is the *Social First Algorithm* (*SFA*). The main idea in *SFA* is to consider users in increasing social distance from the query user. To achieve this, *SFA* expands the social graph around v_q using Dijkstra's algorithm. For every encountered user (i.e., for every vertex popped from Dijkstra's search heap), it also computes the Euclidean distance from u_q and, in turn, the f value. The first k users are placed in the interim result R. For any subsequent user u, if his/her f value is smaller than the current f_k (the k-th largest f score in R), he/she enters the interim result (and evicts from it the user with the maximum f value). The termination condition of *SFA* is based on the fact that the social distance of every un-processed user is lower-bounded by that of the last vertex encountered by Dijkstra's algorithm. Therefore, if v is the last vertex popped from Dijkstra's heap, expression $\theta = \alpha \cdot p(v_q, v)$ lower-bounds the f value of every non-encountered user. Hence, set R is guaranteed to include the correct result when $\theta \geq f_k$, i.e., it is safe for *SFA* to terminate.

Spatial First Approach. *Spatial First Approach* (*SPA*) is another preliminary solution. It processes users in increasing spatial distance from the query user. For this purpose, *SPA* uses an *incremental* nearest neighbor (NN) search in the Euclidean space. To efficiently perform this search, a regular grid index is built on the user locations and a branch-and-bound algorithm is used to retrieve the NNs; this combination is the most suitable for dynamic spatial data kept in main memory [35]. For every encountered user, *SPA* directly calculates their social distance to the query user and inserts them into the interim result R if necessary, similar to *SFA*. If u is the last NN retrieved, expression $\theta = (1-\alpha) \cdot d(u_q, u)$ lower-bounds the f value of every non-encountered user. Hence, set R is guaranteed to be correct when $\theta \geq f_k$.

Although *SFA* and *SPA* are intuitive and simple, they suffer from a major drawback. They are unaware of either the spatial distance or the social distance of the un-processed users, i.e., the value of θ relies solely on either social or spatial information and, therefore, may be too loose. This shortcoming motivates the algorithm described next.

4.2 Twofold Search Approach

In this section we describe an *SSRQ* processing approach which performs concurrently a social and a

spatial search, thus termed *twofold search algorithm* (*TSA*). This twofold search equips *TSA* with two lower bounds for un-processed users (one on their social and the other on their spatial distance from u_q), thus deriving a tighter overall bound on f and alleviating the main drawback of *SFA* and *SPA*. The social search around u_q is performed by Dijkstra's algorithm, similar to *SFA*. The second search is an incremental NN retrieval in the Euclidean space, similar to *SPA*. *TSA* executes in two phases.

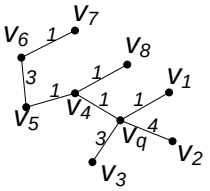
In the first phase, the two searches proceed simultaneously, by iteratively reporting the next closest user in their respective domain, and alternating with each other in a round-robin manner. Whenever the social search is invoked, the encountered user is evaluated, i.e., its f value is computed and checked against the current f_k for potential inclusion into the interim result R. Note that evaluation is fast in this case, because Euclidean distance from u_q is trivial to compute. In contrast, when the spatial search is invoked, the encountered user is either (i) ignored if he/she has already been encountered by the social search or (ii) placed in a candidate set Q. Set Q keeps users that are only partially evaluated, because computing their social distance from q requires expensive processing. The spatial and social search, due to their incremental nature, can be seen as sorted lists (repositories). In this aspect, the first phase of *TSA* follows a hybrid paradigm between TA and NRA (covered in Section 2.4) in that both sorted and random access is possible in the spatial domain, while only sorted accesses are made in the social dimension.

Regarding the termination condition of the first phase, let t_p be the social distance of the last user encountered by the social search, and t_d be the Euclidean distance of the last reported spatial NN. The f value of every user that is not encountered by any of the two searches is lower-bounded by value $\theta = \alpha \cdot t_p + (1-\alpha) \cdot t_d$. The first phase of *TSA* stops when $\theta \geq f_k$. From the definition of bound θ it follows that:

Lemma 1: The final query result may only include users that are either already in the interim result R or in the candidate set Q derived from the first phase of *TSA*.

Based on Lemma 1, the second phase of *TSA* ignores any non-encountered users and aims at evaluating (or disqualifying) candidates in Q. The f value of every candidate is lower-bounded by expression $\theta' = \alpha \cdot t_p + (1-\alpha) \cdot t'_d$ where t'_d is the Euclidean distance of the closest candidate to u_q (in the spatial domain), while t_p is as previously defined. If $\theta' \geq f_k$, *TSA* can terminate. It is obvious that continuing the NN search in the spatial domain cannot affect θ' and would therefore be a waste of computations. Hence, *in the second phase of the algorithm only the social search continues*.

In the second phase, whenever a vertex is output by the Dijkstra search, we perform the following investigation. If the vertex does not belong to Q, it is ignored (by Lemma 1 it cannot be part of the result). If the vertex is in Q, it is removed from Q, it is evaluated and (should its f value be smaller than f_k) it is included into R. In either case, θ' is updated to reflect the new t_p. *TSA* terminates when $\theta' \geq f_k$.



	d	p
u_1	0.1	0.1
u_7	0.1	0.6
u_8	0.6	0.2
u_6	0.7	0.5
u_5	0.7	0.2
u_4	0.8	0.1
u_3	0.9	0.3
u_2	0.9	0.4

Fig. 2. *TSA* example

Algorithm 1 outlines *TSA*. Lines 7-8 include an important detail. In the first phase, it is possible that a candidate is encountered by Euclidean NN search and subsequently discovered by social search too. In this case, it must be removed from Q, since it is fully evaluated. Leaving such candidates in Q would unnecessarily burden the second phase.

Algorithm 1 TSA(G, α, k, u_q)

```
//Input: G: the social graph
//        α: the preference parameter
//        k: the requested number of users
//        u_q: the query user
 1: Initialize Dijkstra and incremental NN search at u_q
 2: Initialize result set R = {}; candidate set Q = {}
 3: while Dijkstra's heap is non-empty do
 4:     Pop next vertex v; en-heap un-visited adj. vertices
 5:     if f(u_q, u) < f_k then
 6:         Update result R, value f_k, and value t_p
 7:     if u ∈ Q then
 8:         Remove v from Q
 9:     Fetch the next nearest neighbor u_nn of u_q
10:     if u_nn was not encountered by Dijkstra search then
11:         Insert u_nn into the candidate set Q
12:     Update value t_d to d(u_q, u_nn)
13:     Set θ = α · t_p + (1 − α) · t_d
14:     if θ ≥ f_k then Break                ▷ End of First Phase
15: Set value t'_d = min_{u∈Q} d(u_q, u)
16: Set θ' = α · t_p + (1 − α) · t'_d
17: while Q is not empty and θ' < f_k do
18:     Fetch the next vertex v from social search
19:     if u ∈ Q then      ▷ u is the user corresponding to v
20:         if f(u_q, u) < f_k then
21:             Update result R and value f_k
22:         Remove u from Q and update value t'_d
23:     Update value t_p
24:     Update θ' = α · t_p + (1 − α) · t'_d
25: Return R
```

TSA **Example:** Figure 2 illustrates eight users $u_1, u_2, ..., u_8$ and u_q. It also includes a table with the Euclidean and social distances of these eight users from u_q, sorted on the former. The order of users in

ascending social distance is $v_1, v_4, v_8, v_5, v_3, v_2, v_6, v_7$. The figure shows only the subgraph of G that is related to our example of processing an *SSRQ* query with $k = 2$ and $\alpha = 0.5$.

TSA first accesses u_1 in the social domain with f value 0.1, and places it into the interim result, i.e., $R = \{u_1\}$. Euclidean NN search fetches u_1, which is ignored because it was previously fully evaluated. *TSA* then discovers u_4 (in the social domain) with f value 0.45 and sets $R = \{u_1, u_4\}$. Next, it encounters u_7 in the Euclidean domain and inserts it into Q. The social search then fetches u_8 with score 0.4 and replaces u_4 in R. The Euclidean search also retrieves u_8, which is ignored. At this stage, $t_p = 0.2$ and $t_d = 0.6$, yielding $\theta = 0.4$. On the other hand, $f_k = 0.4$ which is no larger than θ, and therefore the first phase culminates.

The second phase starts with $Q = \{u_7\}$. Currently, the lower bound θ' (determined by the Euclidean distance of u_7 and t_p) is 0.15, i.e., smaller than f_k. *TSA* continues the social search and iteratively visits new vertices until either u_7 is found (and evaluated) or t_p increases enough so that $\theta' \geq f_k$. In our example, the algorithm terminates when u_7 is encountered by social search, replacing u_8 in the result. *TSA* reports $R = \{u_1, u_7\}$.

TSA with Quick Combine: *Quick Combine* [31] is a popular alternative to round-robin probing for ranked search. This heuristic decides which search (social or spatial) to probe next based on (i) an estimate of how rapidly the distances increase in each domain, and (ii) how large the preference coefficient (α and $(1-\alpha)$) on each domain is. The version of *TSA* that utilizes *Quick Combine* in its first phase is denoted as *TSA-QC*.

TSA with Landmarks: An enhancement to *TSA* is possible if used in conjunction with the landmark approach. Specifically, in a pre-processing stage, a number of vertices in G are chosen as landmarks using the selection technique in [25] and their distances from every other vertex are computed and recorded. Before the second phase of *TSA* starts, we use the landmark information to derive a lower bound of $p(u_q, u)$ for every candidate $u \in Q$. In turn, this produces a lower bound of the candidate's f value (the Euclidean distance of u is already known). If that lower bound is no smaller than f_k, the candidate is eliminated from Q.

5 Aggregate Index Search

Although *TSA* and its landmark-aided version utilize tighter bounds than *SFA*/*SPA*, they may still visit numerous users who are close in the social graph but far away in the spatial domain, and vice versa. The reason is that the two searches are oblivious of each other, and may be accessing completely different users. This motivates a new approach, called *aggregate index search* (*AIS*), which summarizes both social and spatial information into the same index, and runs a unified search on it.

The index is a spatial access method that additionally incorporates (aggregate) social information. Given an index node, we devise a mechanism that provides a lower bound for the f values of all underlying users. This bound is used in a branch-and-bound process to quickly identify users that are close in both domains. The approach incorporates a novel aggregation of landmark information to provide *social summaries* at index nodes, as well as optimized graph access techniques and adaptations of landmarks, tailored to the characteristics of *SSRQ*. We first describe the core of the approach, followed by optimizations in its submodules.

5.1 Aggregate Index and Query Processing

The *AIS* index is a spatial data structure with embedded social information. It could use any spatial access method as a basis (e.g., an R-tree, a k-d-tree, etc). However, we choose a multi-level regular grid because (i) it supports fast location updates [36], [37] and (ii) it facilitates our branch-and-bound *SSRQ* search (the latter being the reason we prefer it over a single-level grid). Each index node is parent to $s \times s$ nodes in the immediately lower level, where s is an integer parameter that determines the partitioning granularity into child nodes. The lowest level contains leaf cells. Each leaf cell C holds the users that lie inside its spatial extent. Figure 3 illustrates an internal index node which is parent to $s \times s$ leaf cells, in an example where $s = 2$. Note that the multi-level grid does not have to be a tree, i.e., it does not necessarily have a root. We may instead keep only a certain number of its lowest levels[1].

The social summaries kept in the index rely on landmark information. *AIS* requires that a set of landmarks is used and that each vertex $v_i \in V$ is associated with a vector including its distances from every landmark. Assuming that there are M landmarks, we denote the distance between vertex v_i and the j-th landmark as m_{ij}. The social summary kept with each cell consists of two vectors, $\hat{m}$ and $\check{m}$, both of length M. Consider first vector $\hat{m}$. Its j-th element is symbolized as $\hat{m}[j]$ and is the maximum path distance between any user in cell C and the j-th landmark. Formally, $\hat{m}[j] = \max_{v_i \in C} m_{ij}$. Similarly, the j-th element of vector $\check{m}$ indicates the minimum path distance between any user in C and the j-th landmark, i.e., $\check{m}[j] = \min_{v_i \in C} m_{ij}$. This information is propagated upwards, setting the social summaries of internal index nodes according to the full set of users they cover.

1. This is the case in our experiments, where keeping the lowest two levels from a three-level hierarchy generally yields favorable performance.

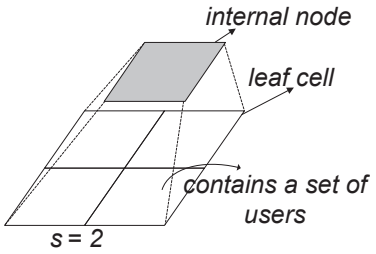
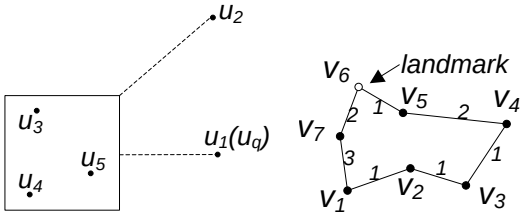


Fig. 3. Internal and leaf cells in *AIS* index



(a) Lower bound of f (b) Social graph

Fig. 4. *AIS* bound example

To enable a branch-and-bound search in the index we need to derive a lower bound on the f values of users in a (internal or leaf) cell C. We first define a lower bound on the spatial domain. Given a query user u_q, we denote as $\check{d}(u_q, C)$ the minimum Euclidean distance between u_q and any point in C. If u_q is inside C, then $\check{d}(u_q, C) = 0$. In all other cases, $\check{d}(u_q, C)$ equals the distance between u_q and the closest point on the boundary of C. For example, in Figure 4(a) the minimum distance between u_1 and the illustrated cell is determined by the horizontal projection line shown dashed. On the other hand, $\check{d}(u_2, C)$ equals the length of the diagonal dashed line.

Regarding the social domain, we show how the aggregate landmark information (vectors $\hat{m}$ and $\check{m}$) can be used to provide a lower bound on $p(v_q, v_i)$ for every $v_i \in C$. Essentially, our technique extends the landmark approach (which was originally proposed for individual vertices) to groups of vertices, i.e., to all vertices under a cell C – to the best of our knowledge, it the first time this is attempted in the literature.

Lemma 2: Given a cell C with social vectors $\hat{m}$ and $\check{m}$, the following formula provides a lower bound for the shortest path distance between any user in C and the query user u_q:

$$\check{p}(u_q, C) = \max_{1 \le j \le M} \begin{cases} \check{m}[j] - m_{qj} & \text{if } m_{qj} < \check{m}[j] \\ m_{qj} - \hat{m}[j] & \text{if } m_{qj} > \hat{m}[j] \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Proof: Consider the j-th landmark and assume that $m_{qj} < \check{m}[j]$. For every $u_i \in C$ the triangular inequality suggests that $p(v_q, v_i) \ge |m_{ij} - m_{qj}|$. Since $m_{qj} < \check{m}[j] \Rightarrow m_{qj} < m_{ij}$, the inequality becomes $p(v_q, v_i) \ge (m_{ij} - m_{qj})$. By definition, $\check{m}[j] \le m_{ij}$ and thus $p(v_q, v_i) \ge (\check{m}[j] - m_{qj})$. As the second part of the inequality is constant for every $u_i \in C$, we deduce that $\min_{u_i \in C} p(v_q, v_i) \ge (\check{m}[j] - m_{qj})$. The case for $m_{qj} > \hat{m}[j]$ is symmetric. On the other hand, when $\check{m}[j] \le m_{qj} \le \hat{m}[j]$, we can derive no lower bound based on the j-th landmark. Finally, we may use the maximum (i.e., tightest) lower bound derived from any landmark as a lower bound for $\min_{u_i \in C} p(v_q, v_i)$. □

Figure 4 illustrates a cell C containing three users and the underlying social graph. Vertex v_6 is chosen as the single landmark ($M = 1$), and the graph distances of v_3, v_4, v_5 are 4, 3, and 1 respectively. Hence, the aggregate information (social summary) kept for this cell is $\hat{m} = 4$ and $\check{m} = 1$. By Formula 2, we can directly derive a lower bound of the social distance between any user in C and $v_q \equiv v_1$ (without accessing the specific social or landmark information of the users in C), i.e., $\check{p}(v_1, C) = 1$, which in this example is as tight as it would be if the exact landmark information of individual users was accessed.

Combing the lower bound $\check{d}(u_q, C)$ for Euclidean distance and $\check{p}(v_q, C)$ for graph distance, we derive a lower bound of the f value for any user in an (internal or leaf) cell C.

Theorem 1: Given a cell C with social vectors $\hat{m}$ and $\check{m}$, the following formula provides a lower bound for the f value of any user in C:

$$\mathit{MINF}(u_q, C) = \alpha \cdot \check{p}(v_q, C) + (1 - \alpha) \cdot \check{d}(u_q, C) \quad (3)$$

Proof: From Lemma 2 and the definition of $\check{d}(u_q, C)$ it follows that $f(u_q, u_i) \ge \mathit{MINF}(u_q, C)$ for each $u_i \in C$. □

Theorem 1 and metric *MINF* pave the way for the *AIS* processing algorithm. The search starts from the top level of the index. All cells in that level are pushed into a min-heap H with keys equal to their *MINF* values. The head of the heap is iteratively popped. Depending on the type of the popped item we distinguish three cases:

- If the item is an internal index node, we push into H all its child nodes with their individual *MINF* values as keys.
- If the item is a leaf cell C, we push into H all the users $u_i \in C$ with key equal to $\alpha \cdot \check{p}(v_q, v_i) + (1 - \alpha) \cdot d(u_q, u_i)$, where $\check{p}(v_q, v_i)$ is the lower bound of social distance $p(v_q, v_i)$ derived from the landmark information of v_i.
- If the item is a user u_i, we compute its exact social distance from v_q (using a submodule described in Section 5.2) and update the interim result R if its f-value is lower than f_k.

The algorithm terminates when the head of the heap has a key larger than or equal to the current f_k. Algorithm 2 summarizes the process.

AIS benefits from combining spatial and social information in the same index and promptly identifies users that lie nearby u_q in both domains. In particular,

Algorithm 2 AIS(G, α, k, u_q)

```
//Input: G: the social graph
//        α: the preference parameter
//        k: the requested number of users
//        u_q: the query user
 1: Initialize an empty min-heap H
 2: Push into H all top-level index nodes with MINF as key
 3: while H is not empty and head's key is less than f_k do
 4:     Pop the head item of H
 5:     if popped item is an internal index node then
 6:         for each child C of the node do
 7:             Push C into H with key MINF(u_q, C)
 8:     else if popped item is a leaf cell C then
 9:         for each user u ∈ C do
10:             Push u into H (key α·p̃(v_q, v)+(1−α)·d(u_q, u))
11:     else if popped item is a user u then
12:         Call a submodule to compute p(v_q, v)
13:         if f(u_q, u) < f_k then
14:             Update result R and value f_k
15: Return R
```

it effectively eliminates nodes, cells and users that are only close in the Euclidean space using the social summaries. On the other hand, for users that are only close in the social graph, it avoids eagerly evaluating them.

The aggregate index supports efficient location updates. When a user u_i moves, the update is dealt with as a deletion in the old cell and an insertion in the new one[2]. We first remove u_i from the user list of the old cell and update the cell's social summaries – if a component in $\hat{m}$ or $\check{m}$ is due to a landmark distance of v_i, the component is recomputed over the remaining users in the cell. Regarding insertion into the new cell, u_i is added to the cell's user list and the landmark distances of v_i are compared against vectors $\hat{m}$ and $\check{m}$. If, say, the j-th landmark distance of v_i is larger than the corresponding component of $\hat{m}$, the latter is set to m_{ij}. Symmetrically, if $m_{ij} < \check{m}[j]$ we update $\check{m}[j]$ to m_{ij}. Should there be an update in the social summary of either the old or the new cell of u_i, it may recursively propagate to upper level nodes in a similar manner. Our index design is primarily concerned with location updates, as the positions of *SN* users change much more frequently/dynamically than the topology of the network. To deal with the latter (i.e., updates in G) batching could be used in conjunction with dynamic shortest path algorithms, so that landmark information can be incrementally maintained [38], [39].

An important remark regards a key principle in designing the index for *AIS*. The index, as described above, partitions the user set according to Euclidean coordinates. Since social summaries are vectors, it is possible to partition the user set (and thus form an index) in the combined social-spatial space. We attempted this approach with little success. We observed that when a space partitioning method is applied to index the combined space, dead space (empty partitions) tends to cripple performance. On the other hand, data partitioning indices cannot effectively balance the relative significance of the two domains (in their bulk-loading and splitting mechanism) without prior knowledge of α and also lead to oblong boxes that compromise performance. Finally, this combined-space approach (be it with a space or data partitioning index) suffers from the dimensionality curse, needing to cope with $M+2$ dimensions. This imposes a serious limitation on the number of landmarks used.

2. Note that if the user moves within his/her current cell, we simply update his/her coordinates; no index maintenance is necessary.

5.2 Graph Search with Computation Sharing

In this section we describe how *AIS* computes social distances for users u popped from its search heap H, i.e., we elaborate on Line 12 of Algorithm 2. First, we decide on the processing paradigm to derive the graph distance. Next, we propose two approaches that enable sharing computations (i.e., reusing information) among the different calls of the submodule for the various evaluated users.

Let u be the user to be evaluated in Line 12, and v be the corresponding vertex in the social graph. *AIS* assumes that landmark information is available for all $v \in V$ in order to build its index. We utilize this landmark information to accelerate the computation of $p(v_q, v)$ too. That is, as described in Section 2.3, an A^* search is applicable – the algorithm proceeds like Dijkstra, but en-heaps encountered vertices with a key incremented by a (landmark-derived) underestimate of their distance to the target vertex. This tends to narrow down the search area of the algorithm. To further enhance performance, instead of a straightforward A^* execution from v_q to the target vertex v, we follow the bidirectional search paradigm [25]. The idea in this paradigm is to concurrently execute two A^* searches: one from v_q to v (called *forward search*) and another from v to v_q (*reverse search*). When the two searches meet, a complete path is derived, which provides a preliminary value for $p(v_q, v)$. This value does not necessarily correspond to the shortest path but facilitates tightening the search; i.e., if the forward or reverse search de-heaps a vertex with key larger than or equal to this distance, the latter can be safely output as the actual graph distance.

The issue is that the above technique is aimed for vertex-to-vertex computations. In *AIS* instead, we need to perform multiple graph distance calculations from the same source v_q to different target vertices. Directly applying the bidirectional approach would perform overlapping searches, i.e., it would unnecessarily repeat part of the work multiple times. Consider for instance Figure 5. Assume that in two consecutive executions of Line 12 in Algorithm 2 we are to obtain the graph distances from v_q to vertices v_{11}

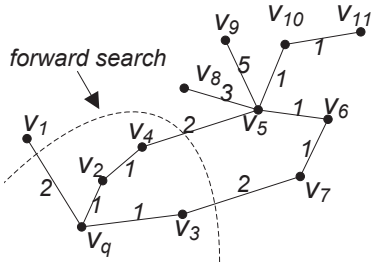


Fig. 5. Path caching

and v_6 respectively. After the first bidirectional search (between v_q and v_{11}), the forward search accesses all vertices inside the dashed boundary. If another bidirectional search is applied between v_q and v_6, forward search starts from scratch and all vertices inside the boundary (i.e., v_2, v_3, v_4) are visited again.

This observation motivates the idea to share computations among different graph distance computations, and therefore save processing time. Before presenting specific techniques to achieve this goal, we must stress that (unlike [25]) our bidirectional approach does not use A^* in both directions. Specifically, while the reverse search is a landmark-based A^* process, for the forward search we employ a plain Dijkstra search, without any aid from landmarks. The reason will become clear shortly.

We describe two complementary computation sharing approaches. The first is conceptually simple.

Distance caching: If the target vertex v was visited by forward search previously, its exact distance has already been computed and can be reported directly. Continuing the example in Figure 5, if v_4 happens to be the next target vertex, its distance from v_q is already known because the forward search between v_q and v_{11} has previously visited it (the distance of any vertex popped from the Dijkstra heap is immediately derived). Similarly, if v belongs to a previously reported shortest path, its distance from v_q is also readily available (when a vertex belongs to the shortest path between a source and a target, its distance from either is directly deduced).

Forward heap caching: The second technique reuses the search heap of the forward search. Instead of terminating forward search and re-invoking it from scratch for every target vertex, we maintain its heap contents and re-use them between runs. That is, essentially the forward search only pauses when the graph distance to a target vertex v is found and its state (i.e., its search heap) is maintained. When the distance of the next target v' is to be computed, the forward search resumes from the point it stopped, using the already populated heap.

Note that for this optimization to be possible, forward search must be implemented as a Dijkstra process. The rationale is that in Dijkstra's algorithm the keys used in the search heap are irrelevant to the target vertex, and this exactly is the fact that enables reusing the heap for different target vertices. In an A^* implementation of forward search, the heap keys would be incremented by (landmark-derived) distance bounds that depend on the specific target vertex each time, making the heap useless for different target vertices.

The distance computation submodule of *AIS* with all optimizations is outlined by procedure *GraphDist* (Algorithm 3). H_f is the search heap of forward search. T is a table including previously computed shortest paths. H_f and T are global variables, i.e., they are retained between the calls of *GraphDist* and discarded only when *AIS* (the calling process) terminates.

Algorithm 3 GraphDist(G, v_q, v, H_f, T)

```
//Input: G: the social graph
//        v_q: the (vertex corresponding to the) query user
//        v: target vertex to compute the graph distance to
//        H_f: the min-heap of forward search
//        T: set of all previously computed shortest paths
1: if v was previously visited by forward search then
2:     Return the stored distance of v
3: else if v appears in any path in T then
4:     Return the stored distance of v
5: Initialize MinDist = +∞ and ShortestPath = {}
6: Initialize an A* process at v for the reverse search
7: while MinDist > head's key in heap of rev. search do
8:     Fetch next vertex v_f from forward search (from H_f)
9:     if v_f was previously visited by reverse search then
10:        if p(v_q, v_f) + p(v_f, v) < MinDist then
11:            Set MinDist = p(v_q, v_f) + p(v_f, v)
12:            Update ShortestPath accordingly
13:    Fetch next vertex v_r from reverse search
14:    if v_r was previously visited by forward search then
15:        if p(v_q, v_r) + p(v_r, v) < MinDist then
16:            Set MinDist = p(v_q, v_r) + p(v_r, v)
17:            Update ShortestPath accordingly
18:            Do not push nodes adj. to v_r into rev. heap
19: Store ShortestPath in T
20: Return MinDist
```

5.3 Improving on Landmark Lower Bounds

Referring to the general *AIS* algorithm, as described in Section 5.1, vertices are evaluated in an order dictated by a lower bound of their f values (see Line 10 in Algorithm 2). This lower bound is derived in part by landmark estimates of the social distance between v_q and the vertices. It is a known fact that landmarks often produce very loose lower bounds, which may lead *AIS* to evaluate target vertices that in reality lie too far from v_q in the social graph.

Consider for instance Figure 6, and assume that v_5 is used as the landmark. Vertices v_q and v_7 are almost equi-distant from the landmark, yielding a lowed bound $\check{p}(v_q, v_7) = 1$. This is a large underestimate of the actual distance and may lead in evaluating v_7 (via expensive search in G), although it is actually

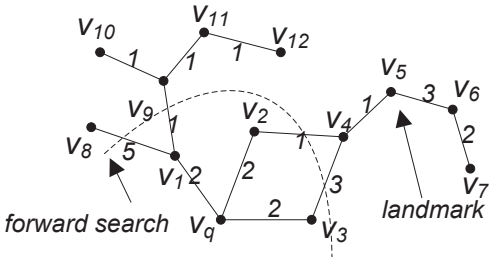


Fig. 6. Delayed evaluation example

too far from v_q in the social space. Using a large number of landmarks could reduce the occurrence of wide underestimates, but it is not a panacea; as we show in the experiments, using many landmarks could seriously harm overall performance.

Fortunately, the nature of our algorithm allows for information sharing that may alleviate this problem. Specifically, as *AIS* evaluates more users, the forward search in its bidirectional submodule also proceeds. Let β be the key (graph distance) of the last vertex popped in forward search. If the target vertex v in Line 11 of Algorithm 2 has not been visited by the forward search before, we are sure that its social distance from v_q is at least β, i.e., β may serve as a lower bound of $p(v_q, v)$ which, actually, might be tighter (larger) than the landmark-based $\check{p}(v_q, v)$. If that is the case, we derive a new (larger) lower bound for $f(v_q, v)$ (that is $\alpha \cdot \beta + (1-\alpha) \cdot d(u_q, u)$) and push v back into the *AIS* heap with the new bound as the key. This may postpone the premature evaluation of vertices due to wide landmark underestimates. We refer to this technique as *delayed evaluation strategy*.

Consider again the example in Figure 6, and assume that when v_7 is popped from the heap of *AIS* the β value is 2. This means that the forward search (due to the evaluation of previous target vertices) has reached up to the boundary shown dashed. Instead of directly computing the actual graph distance of v_7 (in Line 12 of Algorithm 2), we detect that its landmark distance is looser than β and re-insert it into the *AIS* heap with an updated key based on β.

Note that a vertex might be re-inserted into the *AIS* heap multiple times before it is actually evaluated. This is the case when a re-inserted vertex is popped anew, but the β value has meanwhile further increased, leading to an even tighter lower bound for its f value. In this situation, the vertex is pushed into H again with a new key. To incorporate the delayed evaluation strategy we need to add the following instructions right after Line 11 in Algorithm 2:

1: **if** key of u in H is less than $\alpha \cdot \beta + (1-\alpha) \cdot d(u_q, u)$ **then**
2: **if** v not visited by forw. search nor exists in T **then**
3: Push u back into H with key $\alpha \cdot \beta + (1-\alpha) \cdot d(u_q, u)$
4: Go to Line 3 (of Algorithm 2)

The second condition is to avoid re-inserting a vertex whose graph distance is readily available (because it has been visited by forward search, or because it belongs to an already computed shortest path).

5.4 Graph Distance Pre-computation

Given that social search dominates the processing cost (in all approaches), pre-computing social distances between vertices could possibly improve performance. Materializing all-pair social distances requires a prohibitive amount of storage; for the *Foursquare* graph in our experiments, which contains around 2 million users, we need roughly 16 TB to store all-pair distances. To alleviate this problem, we could instead materialize for each user the distances of the t socially closest vertices. To utilize the pre-computation, we replace *SFA*'s Dijkstra component with the pre-computed distance list. In case the algorithm exhausts the list of t social neighbors (without terminating), it falls back to our best method, *AIS*. Note that pre-computation is applicable to SPA and TSA as well, but with limited success, because these algorithms may encounter a socially distant candidate (outside the pre-computed list) very early in their execution.

6 Experimental Evaluation

In this section we experimentally evaluate the *SSRQ* techniques proposed in the paper. All methods were implemented in C++ and the experiments conducted on an Intel Core2Duo 2.66GHz CPU machine with 8 GB memory, running on Ubuntu 10.04.

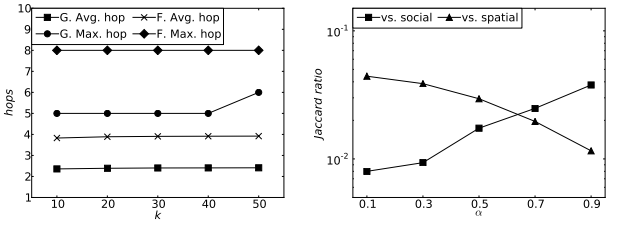
We use two real datasets, *Gowalla* and *Foursquare*. Table 2 provides some of their characteristics; the last column indicates their average vertex degree. *Gowalla*, obtained from snap.stanford.edu, contains 196K users. *Foursquare*, used in [40], [41], contains 1.88M users. Due to privacy constraints, the location records for some users are unavailable. Thus, we only have access to the historical positions of 54.4% of users in *Gowalla* and those of 60.3% of users in *Foursquare*.[3] From the locations available for a user, we assign him/her the one with the highest frequency of visits.

TABLE 2
Data Statistics

Name	$\|V\|$	$\|E\|$	# locations	Deg.
Gowalla	196,590	1,900,654	107,092	9.7
Foursquare	1,880,405	17,838,254	1,133,936	9.5

Neither of the social networks has explicit information about the edge weights. Based on a common methodology [2], [42], we derive this information from the degrees of vertices incident to the edges. Intuitively, the more the friends of a user, the looser the connection to them, i.e., the larger the edge weight.

3. Users with no available location are considered infinitely far away from any other user.



(a) Hop statistics (b) *SSRQ* vs. social/spatial NNs

Fig. 7. Insights into the nature of *SSRQ* query

Thus, edge weights are set proportionally to the product of degrees of the vertices (users) they connect, i.e., the weight of edge (v_i, v_j) is set to $\frac{deg(v_i) \cdot deg(v_j)}{max\_degree^2}$, where $deg(v_i)$ and $deg(v_j)$ are the degrees of vertices v_i and v_j respectively, and max_degree is the maximum vertex degree in the social graph.

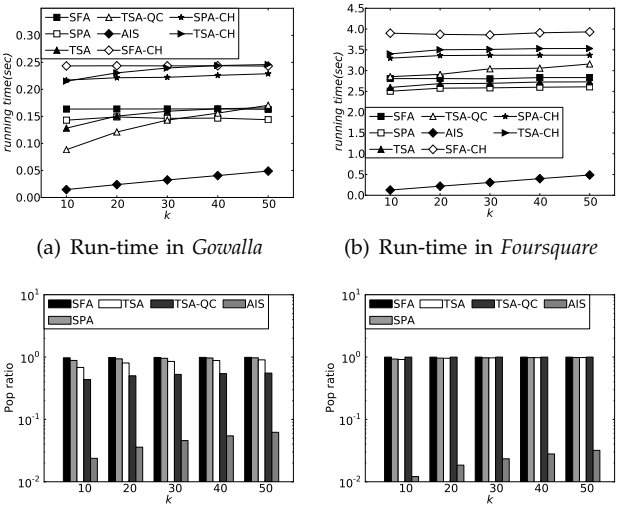
Table 3 includes the tested value ranges for the query and system parameters in our setup. In each experiment, unless otherwise stated, the parameters are set to the default values shown in the table. Data and indices for all methods are kept in memory. The main performance factor in our evaluation is *run-time*, i.e., query processing cost. We also report the *pop ratio*, computed as $\frac{|V_{pop}|}{|V|}$, where $|V_{pop}|$ is the number of vertices popped from the search heaps of the methods. Importantly, the pop ratio measurements are also indicative of performance (specifically, I/O cost) in an alternative setting where the social graph is stored on the disk. Every reported measurement is the average across 1,000 *SSRQ* random queries.

TABLE 3
Query and System Parameters

Parameter	Default	Range
size of result k	30	10, 20, 30, 40, 50
preference parameter α	0.3	0.1, 0.3, 0.5, 0.7, 0.9
grid granularity s	10	5, 10, 15, 20, 25

We first study our data and the nature of *SSRQ*. In Figure 7(a) we record the number of hops (away from v_q) where the furthest *SSRQ* result is found. We plot the AVG and MAX of these numbers (across the 1,000 queries run for each k value tested). Prefix "F." corresponds to *Foursquare* and "G." to *Gowalla*. We see that results may lie several hops away from v_q, in some cases reaching up to 8 hops.

In Figure 7(b) we investigate the similarity (i) between the *SSRQ* result and the k Euclidean NNs of u_q and (ii) between the *SSRQ* result and the k socially closest users to v_q. In either case, we compute the Jaccard ratio, a standard measure of set similarity [43]. Given two sets, it is defined as the cardinality of their intersection divided by the cardinality of their union; its domain is 0 (unrelated sets) to 1 (identical sets). We plot results for different α values in *Foursquare*, i.e., we vary the relative importance of spatial and social proximity. The Jaccard ratio is below 0.1 in all cases, indicating that the nature of *SSRQ* is very different



(a) Run-time in *Gowalla* (b) Run-time in *Foursquare*

(c) Pop ratio in *Gowalla* (d) Pop ratio in *Foursquare*

Fig. 8. Effect of k

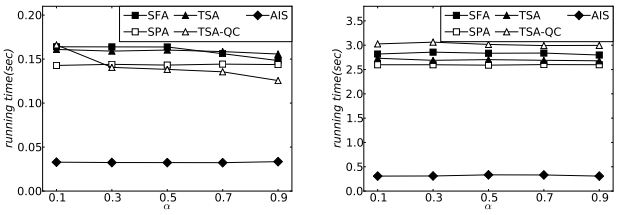
from either social or spatial NN search. Results in *Gowalla* are similar and omitted for brevity.

Next we compare the different *SSRQ* approaches. *SFA* and *SPA* are described in Section 4. *TSA* is the landmark-aided version of the algorithm in Section 4.2 (we disregard its non-landmark counterpart because it consistently performs worse). *TSA-QC* is the *Quick Combine* version of *TSA*. *AIS* is the best-performing version of *aggregate index search* from Section 5 (its different flavors are evaluated later). We finetuned the landmark-based methods with respect to M (number of landmarks) and set it to 8.

Figure 8 investigates performance for different values of k. Figures 8(a) and 8(b) show that processing in *Gowalla* is faster than *Foursquare* – the reason is that *SSRQ* search in *Gowalla*, regardless of the algorithm chosen, generally reaches fewer users than in *Foursquare*, as shown in Figure 7(a). The run-time increases with k, because the search area in both the social and the spatial domain expands for larger k.

Just for this experiment, in the run-time charts we include variants SFA-CH, SPA-CH and TSA-CH (of SFA, SPA, TSA) where we replaced the Dijkstra-based social distance computation module with the state-of-the-art pre-computation-based technique for shortest path computation, CH, from [44]. These variants are slower than vanilla SFA/SPA/TSA, because (i) CH is better suited to low-degree graphs (such as planar graphs) and (ii) in vanilla methods, shortest paths are produced incrementally (they all have v_q as source), thus essentially sharing/reusing computations.

Turning to the relative performance of our algorithms, Figures 8(c) and 8(d) verify that *SFA* and *SPA* process more vertices than *TSA*, due to their looser termination conditions. However, the difference in run-time is not as wide as in pop ratio, the reason being *TSA*'s overhead in maintaining two bounds (one spatial and one social). *TSA-QC* performs better



(a) Run-time in *Gowalla* (b) Run-time in *Foursquare*

Fig. 9. Effect of α

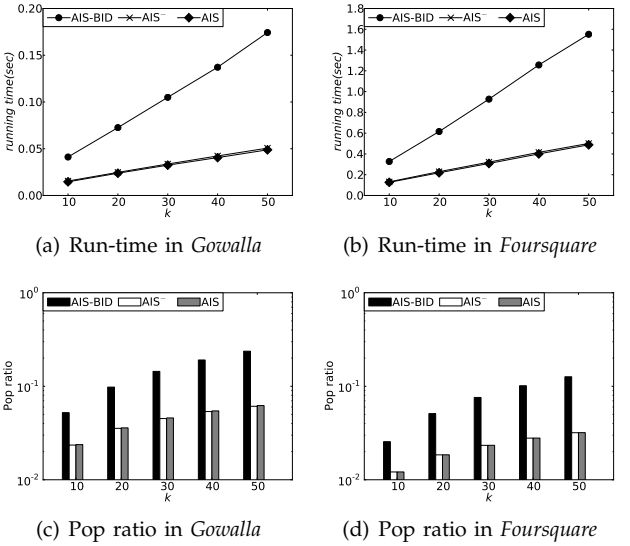
than *TSA* in *Gowalla* when k is small but is slower than *TSA* in *Foursquare* in all settings. We suspect that this is due to the different social/spatial distributions in the two datasets. *AIS* visits fewer than 6% and 3% of the vertices in *Gowalla* and *Foursquare* respectively, while the other approaches visit more than 90% in most cases. This demonstrates that the *aggregate index search* paradigm vastly reduces the number of expanded vertices (especially in large *SN*s), which implies significantly shorter processing time.

In Figure 9 we test different values of α, i.e., different weighing of social versus spatial proximity. *SFA* examines vertices in increasing social distance order, which implies that for large α the first few processed vertices are highly likely to already produce the result. *TSA* and *TSA-QC* are also more socially-led (than spatially), since their second phase relies entirely on graph search, thus benefiting from a large α. *SPA*, on the other hand, is spatially-led and hence its performance worsens with α, albeit only slightly. Importantly, our most advanced method, *AIS*, is robust to α and retains its clear lead over alternatives.

Aggregate index search provides a flexible framework, which can be equipped with different techniques and optimizations. Here we evaluate its three most representative versions: (i) *AIS-BID* is a direct implementation of Algorithm 2 using the bidirectional search in [25] for graph distance computations and no other optimization; (ii) *AIS*$^-$ uses all optimizations *except the delayed evaluation strategy*; and (iii) *AIS* uses all optimizations.

In Figure 10 we compare these algorithms for various values of k. The behavior of *AIS-BID* demonstrates that, although the search technique proposed in [25] is more efficient than other flavors of bidirectional search, it is unable to yield favorable performance without our further enhancements. This fact is also supported by the pop ratio charts. The comparison between *AIS* and *AIS*$^-$ reveals that the delayed evaluation mechanism improves performance, albeit to a moderate degree.

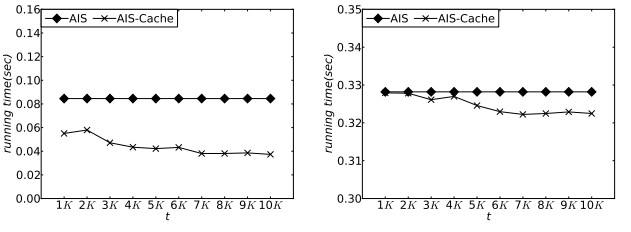
In Figure 11 we evaluate the pre-computation technique (from Section 5.4) against *AIS*. We present runtime versus t, i.e., versus the number of cached social neighbors per user. Pre-computation yields minor improvements in the larger graph (*Foursquare*) but significant in the smaller one (*Gowalla*). The reason



(a) Run-time in *Gowalla* (b) Run-time in *Foursquare*

(c) Pop ratio in *Gowalla* (d) Pop ratio in *Foursquare*

Fig. 10. Effect of k on AIS versions

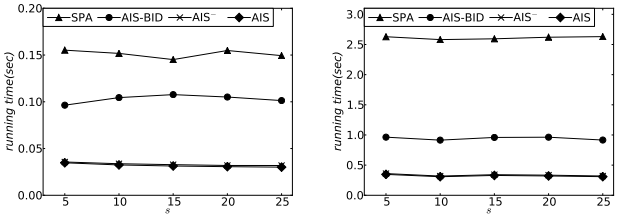


(a) Run-time in Gowalla (b) Run-time in Foursquare

Fig. 11. Effect of pre-computation

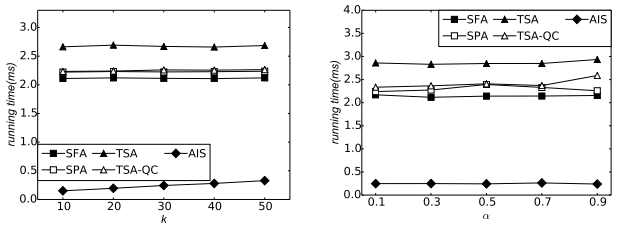
is that, as shown in Figure 7(a), in *Foursquare* search expands more hops away from v_q, thus being more likely to reach a vertex outside the cache.

In Figure 12, we measure the effect of s, i.e., the granularity of the grid index, on *SPA*, *AIS-BID*, *AIS*$^-$ and *AIS*. Recall that a larger s implies more cells with smaller size each. This parameter affects performance in two conflicting ways: (i) as s grows, more cells lie in the vicinity of the query user and therefore more computations are needed to calculate distance bounds for them; (ii) on the other hand, smaller grid cells provide more accurate summaries (be them Euclidean or social) about the underlying users, and increase the effectiveness of pruning. Value $s = 10$ strikes a favorable balance between these factors, although the methods are not very sensitive to it.



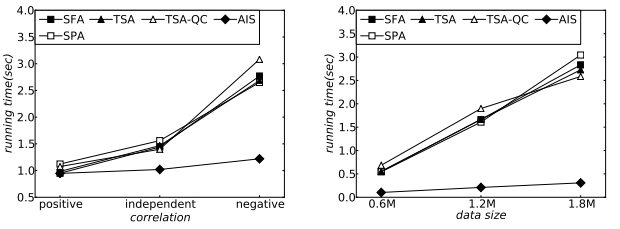
(a) Run-time in *Gowalla* (b) Run-time in *Foursquare*

Fig. 12. Effect of grid granularity



(a) Effect of k (b) Effect of α

Fig. 13. Experiments with *Twitter*



(a) Correlation (b) Data size

Fig. 14. Experiments with synthetic data

For generality, in Figure 13 we use a real dataset, *Twitter*, with higher average degree than our default datasets (its average degree is 57.7). It contains 124K Twitter users in Singapore who made geo-tagged tweets in 2013; a user's location is derived from his/her latest tweet. The charts (versus k and α) show similar trends to our default datasets. A difference is that the run-time increases less sharply with k, because the larger degree implies that more candidates (users) are reachable with fewer hops from u_q.

Finally, we generate synthetic data to examine parameters we cannot control in the real *SN*s. In Figure 14(a), we generate data with different correlations between the social and spatial distances. We use the social distances derived from *Foursquare*, but assign to users artificial locations as follows. For each v_q, we generate the spatial distance of user u from u_q by formula $\bar{d} = \rho \cdot p(v_q, v) + \epsilon$, where ϵ is a random number in range $[-0.15, 0.15]$ and ρ is 1 (for positive-correlation dataset) or -1 (for negative-correlation dataset). Based on the generated $\bar{d}$ (normalized in the [0,1] range), we place the user at a random point on the circle with radius $\bar{d}$ from u_q. We also generate a third dataset, where the spatial locations of users are randomly permuted, so as to create a dataset with independent correlation between social and spatial proximity.

All algorithms require the shortest time when data are positively correlated and the longest when they are negatively correlated. In the positive correlation case, users that are socially near v_q tend to also lie close by in the Euclidean space. Hence, search encounters the top-k users early on and terminates quicker. The situation is reversed for negatively correlated data, because socially near users are spatially far, and vice versa, implying that the top-k users tend to lie far from the query in either of the two domains, if not in both. *AIS* is the method of choice in all cases, furthermore demonstrating robustness to the type of correlation between spatial and social distance.

In Figure 14(b), we show performance for different *SN* sizes. Starting with *Foursquare* as a basis, we extracted from it *SN*s of different sizes using the structure-preserving *Forest Fire Sampling* technique [45]. As the number of *SN* vertices is tripled from 0.6M to 1.8M, the running time of all algorithms increases almost linearly, with *AIS* scaling much more gracefully than competitors.

7 Conclusion

We study a query type that captures proximity in the combined social-spatial domain. Our most efficient algorithm relies on an aggregate index that supports estimates of combined proximity. Experiments on actual social networks demonstrate that it is highly scalable and robust. A direction for future work is joint social and spatial processing on networks stored in a distributed manner.

Acknowledgments

This work was funded by research grant 14-C220-SMU-004 from the Singapore Management University Office of Research under the Singapore Ministry of Education Academic Research Funding Tier 1 Grant, and by grant HKU 715413E from Hong Kong RGC. The authors would also like to thank Dr. Mohamed Sarwat for providing datasets and suggestions.

References

[1] M. Helft, "Bing taps facebook data for fight with google," http://bits.blogs.nytimes.com/2011/05/16/, May 2011.
[2] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003, pp. 137–146.
[3] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. C. Reis, and B. A. Ribeiro-Neto, "Efficient search ranking in social networks," in *CIKM*, 2007, pp. 563–572.
[4] P. Yin, W.-C. Lee, and K. C. K. Lee, "On top-k social web search," in *CIKM*, 2010, pp. 1313–1316.
[5] D. Papadias, M. L. Yiu, N. Mamoulis, and Y. Tao, "Nearest neighbor queries in network databases," in *Encyclopedia of GIS*, 2008, pp. 772–776.
[6] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *KDD*, 2002, pp. 61–70.
[7] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *KES (3)*, 2008, pp. 67–75.
[8] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.
[9] A. Goyal, F. Bonchi, and L. Lakshmanan, "A data-based approach to social influence maximization," *PVLDB*, vol. 5, no. 1, pp. 73–84, 2011.
[10] M. E. J. Newman, "Clustering and preferential attachment in growing networks," in *Physical Review Letters E*, 2001.
[11] D. Liben-Nowell and J. M. Kleinberg, "The link-prediction problem for social networks," *JASIST*, vol. 58, no. 7, pp. 1019–1031, 2007.
[12] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 1483–1484.
[13] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.

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

[15] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *SIGMOD Conference*, 2011, pp. 349–360.

[16] S. Scellato, A. Noulas, and C. Mascolo, "Exploiting place features in link prediction on location-based social networks," in *KDD*, 2011, pp. 1046–1054.

[17] M. Ye, P. Yin, W.-C. Lee, and D. L. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation," in *SIGIR*, 2011, pp. 325–334.

[18] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *PVLDB*, vol. 6, no. 10, pp. 913–924, 2013.

[19] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *KDD*, 2011, pp. 1082–1090.

[20] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler, "Buddy tracking - efficient proximity detection among mobile friends," *Pervasive and Mobile Computing*, vol. 3, no. 5, pp. 489–511, 2007.

[21] G. Treu, T. Wilder, and A. Kupper, "Efcient proximity detection among mobile targets with dead reckoning," in *MOBIWAC*, 2006, pp. 75–83.

[22] M. L. Yiu, L. H. U, S. Saltenis, and K. Tzoumas, "Efficient proximity detection among mobile users via self-tuning policies," *PVLDB*, vol. 3, no. 1, pp. 985–996, 2010.

[23] Z. Xu and H.-A. Jacobsen, "Adaptive location constraint processing," in *SIGMOD Conference*, 2007, pp. 581–592.

[24] J. Bao, M. F. Mokbel, and C.-Y. Chow, "Geofeed: A location aware news feed system," in *ICDE*, 2012, pp. 54–65.

[25] A. V. Goldberg and C. Harrelson, "Computing the shortest path: *A** search meets graph theory," in *SODA*, 2005, pp. 156–165.

[26] H.-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt, "Hierarchical graph embedding for efficient query processing in very large traffic networks," in *SSDBM*, 2008, pp. 150–167.

[27] J. Matousek, "On the distortion required for embedding finite metric spaces into normed spaces," *Israel Journal of Mathematics*, vol. 93, no. 1, pp. 333–344, 1996.

[28] M. Thorup and U. Zwick, "Approximate distance oracles," *J. ACM*, vol. 52, no. 1, pp. 1–24, 2005.

[29] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy, "A sketch-based distance oracle for web-scale graphs," in *WSDM*, 2010, pp. 401–410.

[30] J. Cheng and J. X. Yu, "On-line exact shortest distance query processing," in *EDBT*, 2009, pp. 481–492.

[31] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-*k* query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, 2008.

[32] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, 2003.

[33] N. Bruno, L. Gravano, and A. Marian, "Evaluating top-k queries over web-accessible databases," in *ICDE*, 2002, pp. 369–380.

[34] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *PVLDB*, vol. 3, no. 1, pp. 373–384, 2010.

[35] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *SIGMOD Conference*, 2005, pp. 634–645.

[36] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch, "Main memory evaluation of monitoring queries over moving objects," *Distributed and Parallel Databases*, vol. 15, no. 2, pp. 117–135, 2004.

[37] M. F. Mokbel, X. Xiong, and W. G. Aref, "Sina: Scalable incremental processing of continuous queries in spatio-temporal databases," in *SIGMOD Conference*, 2004, pp. 623–634.

[38] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Trans. Netw.*, vol. 8, no. 6, pp. 734–746, 2000.

[39] E. P. F. Chan and Y. Yang, "Shortest path tree computation in dynamic graphs," *IEEE Trans. Computers*, vol. 58, no. 4, pp. 541–557, 2009.

[40] M. Sarwat, J. Bao, A. Eldawy, J. J. Levandoski, A. Magdy, and M. F. Mokbel, "Sindbad: a location-based social networking system," in *SIGMOD Conference*, 2012, pp. 649–652.

[41] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel, "Lars: A location-aware recommender system," in *ICDE*, 2012, pp. 450–461.

[42] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *KDD*, 2010, pp. 1029–1038.

[43] M. Levandowsky and D. Winter, "Distance between sets," *Nature*, vol. 234, pp. 34–35, 1971.

[44] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou, "Shortest path and distance queries on road networks: An experimental evaluation," *PVLDB*, vol. 5, no. 5, pp. 406–417, 2012.

[45] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *KDD*, 2006, pp. 631–636.

Kyriakos Mouratidis is an Associate Professor at Singapore Management University. He received his BSc from the Aristotle University of Thessaloniki, Greece, and his PhD from the Hong Kong University of Science and Technology. His main research area is spatial databases, with a focus on continuous queries, road networks and spatial optimization problems. He has also worked on preference queries, wireless broadcasting systems, and certain database privacy topics.

Jing Li received the BSc degree in computer science and engineering from Nanjing University, and the PhD degree from the Department of Computer Science, University of Hong Kong. His research interests include data privacy, query processing in spatio-temporal databases, graph databases, and data mining in textual data streams.

Yu Tang is a MPhil student at the Department of Computer Science, The University of Hong Kong. His research interests are mainly in theoretical and system aspects of databases and data management.

Nikos Mamoulis received a diploma in Computer Engineering and Informatics from the University of Patras, Greece, and a PhD in Computer Science from the Hong Kong University of Science and Technology. He is currently an Associate Professor at the Department of Computer Science, University of Hong Kong. He has been involved in the organization of several workshops/conferences, and serves as associate editor for IEEE TKDE and VLDB Journal.