Geo-Social Skyline Queries

Tobias Emrich[1], Maximilian Franzke[1], Nikos Mamoulis[2], Matthias Renz[1], and Andreas Züfle[1]

[1] Ludwig-Maximilians-Universität München
emrich, franzke, renz, zuefle@dbs.ifi.lmu.de,
[2] The Hong Kong University
nikos@cs.hku.hk

Abstract. By leveraging the capabilities of modern GPS-equipped mobile devices providing social-networking services, the interest in developing advanced services that combine location-based services with social networking services is growing drastically. Based on geo-social networks that couple personal location information with personal social context information, such services are facilitated by geo-social queries that extract useful information combining social relationships and current locations of the users. In this paper, we tackle the problem of geo-social skyline queries, a problem that has not been addressed so far. Given a set of persons $\mathcal{D}$ connected in a social network *SN* with information about their current location, a geo-social skyline query reports for a given user $U \in \mathcal{D}$ and a given location P (not necessarily the location of the user) the pareto-optimal set of persons who are close to P and closely connected to U in *SN*. We measure the social connectivity between users using the widely adoted, but very expensive Random Walk with Restart method (RWR) to obtain the social distance between users in the social network. We propose an efficient solution by showing how the RWR-distance can be bounded efficiently and effectively in order to identify true hits and true drops early. Our experimental evaluation shows that our presented pruning techniques allow to vastly reduce the number of objects for which a more exact social distance has to be computed, by using our proposed bounds only.

1 Introduction

In real life, we are connected to people. Some of these connections may be stronger than others. For example, for some individual the strength of a social connection may monotonically decrease from their partner, family, friends, colleagues, and neighbours to strangers. Social connections (or their lack of) define social networks that extend further than just a person's acquaintances: There are friends of friends, stepmothers and contractors that stand in an indirect relation to a person; eventually reaching every person in the network. Such social networks are used, consciously or unconsciously, by everybody to find amiable people for a plethora of reasons: To find people to join a common event such as a concert or to have a drink together or to find help, such as a handyman or an expert in a specific domain. Yet, the person with the strongest social connection to may not be the proper choice due to non-social aspects. This person may not be able to help with a specific problem due to lack of expertise, or the person may simply be too far away to join. For example, in a scenario where your car has

broken down, you are likely to accept the help of a stranger. When you are travelling, for example visiting a conference in Bali, the people you are strongly connected to are likely to be too far away to join you for a drink.

Also, another interesting problem arises when travelling and you need a place to stay for the night. Assume you do not want to book a hotel but rather sleep at someone's home (aka "couchsurfing"). Of course it's most convenient if you have a strong social connection to someone close to your destination, but the farther you travel from home, the less likely this becomes, as most of your acquaintances are usually spatially close to you [22]. Trying to find the person best suited to accommodate you, you face the following trade-off: Rather stay with someone you have less social connections to but can provide shelter close to your destination or you accept longer transfer times and choose to stay with someone more familiar. Since this trade-off depends on personal preferences, a skyline query is suitable: By performing a skyline query, a user obtains a list of people, with each person's attributes being a pareto-optimum between social distance to the user and spatial distance to their destination. Driven by such applications, there is a new trend of novel services enabled by geo-social networks coupling social network functionality with location-based services. A geo-social network is a graph where nodes represent users with information about their current location and edges correspond to friendship relations between the users [3]. User locations are typically provided by modern GPS-equipped mobile devices enabling check-in functionality, i.e. the user is able to publish his current location by "checking in" at some place, like a restaurant or a shop. Example applications based on geo-social networks are Foursquare and novel editions of Facebook and Twitter that adopted the check-in functionality recently.

Extracting useful information out of geo-social networks by means of geo-social queries taking both the social relationships and the (current) location of users into account is a new and challenging problem, first appraoches have been introduced recently [3]. In this paper we tackle the geo-social skyline query problem. This is the first approach for this problem. Given a set of persons $\mathcal{D}$ connected in a social network *SN* with information about their current location a geo-social skyline query reports for a given user $U \in \mathcal{D}$ and a given location P (not necessarily the location of the user) the pareto-optimal set of persons who are close to P and closely connected to U in *SN*. In particular we present and study initial approaches to compute the geo-social skyline efficiently which is challenging as we apply the very expensive Random Walk with Restart distance to measure the social connectivity between users in the social network. The basic idea of our approach is that for skyline-queries it is not necessary to calculate exact social distances to all users, which is very expensive. In a nutshell, we efficiently determine lower and upper distance bounds used to identify the skyline. The bounds are iteratively refined on demand allowing early termination of the refinement process.

2 Problem Definition

In the following we will define the problem of geo-social skyline query tackled in this paper. Furthermore, we discuss methods for measuring the social similarity in social networks which we apply to compute the geo-social skyline.

2.1 Geo-Social Skyline Query

The problem of answering a geo-social skyline query is formally defined as follows.

Definition 1. *Geo-Social Skyline Query (GSSQ)*
Let $\mathcal{D}$ be a geo-social database, consisting of a set of users U, where each user $u \in U$ is associated with a geo-location $u.loc \in \mathcal{R}^2$. Let $S = (\text{<U>}, \text{<L>})$ be a social network consisting of a set <U> of vertices corresponding to users, and a set <L> of weighted links between users. Furthermore, let $dist_{geo} : U \times U \rightarrow \mathcal{R}$ be a geo-spatial distance measure, and let $dist_{soc} : U \times U \rightarrow \mathcal{R}$ be a social distance measure. A geo-social skyline query (GSSQ) returns, for a given geo-location $q_{geo} \in \mathcal{R}^2$ and a given user $q_{soc} \in U$ the set of users $u \in GSSQ(q_{geo}, q_{soc}) \subseteq U$, s.t.

$$u \in \textit{GSSQ}(q_{geo}, q_{soc}) \Leftrightarrow$$
$$\neg\exists u' \in U : dist_{soc}(q_{soc}, u') < dist_{soc}(q_{soc}, u) \wedge dist_{geo}(q_{geo}, u') < dist_{geo}(q_{geo}, u)$$

Informally, a *GSSQ* returns, for a given geo-location q_{geo} and a given user q_{soc}, the set of users such that for each user $u \in \textit{GSSQ}(q_{\text{geo}}, q_{\text{soc}})$ there exists no other user $u' \in U$ such that user u has both a larger spatial distance to q_{geo} and a larger social distance to q_{soc} than user u'.

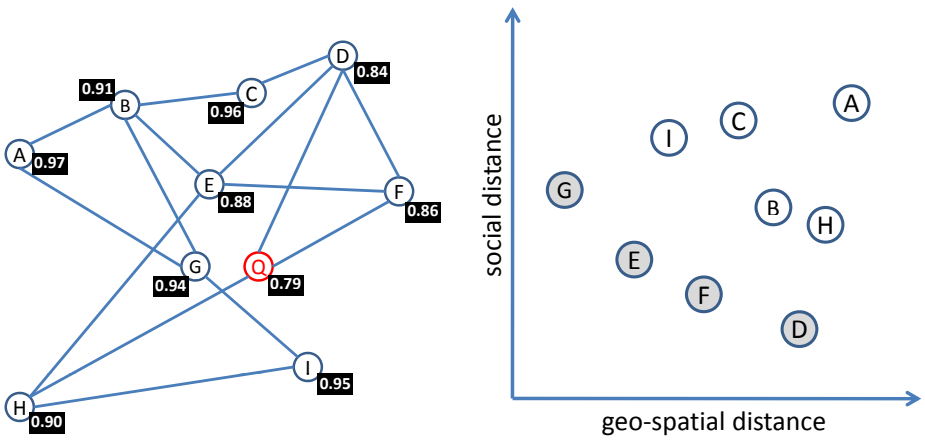
By design, we exclude the query node from the result set: In the foreseeable applications including oneself in the result gives no additional information gain, but may actually prune and therefore exclude other nodes from the result. This design decision is without loss of generality; depending on application details the query node itself may be allowed to be a valid result as well.

Yet, we haven't specified the distance measures involved in the *GSSQ*. In the following, we discuss the notion of similarity in the two domains in involved in our query problem, the geo-spatial domain and the social domain and introduce the two similarity distance measures, the geo-spatial distance and social distance, respectively.

An example of a *GSS*-query is shown in Figure 1. The social network, the geo-location of each user and the social distance of each user to Q (numbers in black boxes) are shown in Figure 1(a). The resulting skyline-space is illustrated in Figure 1(b), with users G, E, F and D being the result of the query. The semantics of the result in this example is also very interesting. Ranked by distance we get the following result: user G is more or less a stranger but has the closest distance, user E is a friend-of-a-friend, user F is a close friend and user D is the best friend which however has a very large distance. The result thus gives the user Q free choice in the trade-off between importance of social and spatial distance.

2.2 Geo-Spatial Similarity

For the geo-spatial distance, instead of using the common Euclidean distance, we decided to use the geodetic distance which is the shortest distance between two points on Earth along the Earth's surface (simplified as a sphere). Given that locations are specified by longitude and latitude coordinates, this distance measure is more adequate, in



(a) Geo-social network (numbers represent social distance to Q)

(b) Geo-social skyline

Fig. 1. The geo-social skyline of a geo-social network.

particular for long distance measures, than the Euclidean distance and is depending on the Earth's radius r (approx. 6371km). The geo-spatial distance is defined as:

$$dist_{\text{geo}}(u_1^{\text{lat}}, u_1^{\text{long}}, u_2^{\text{lat}}, u_2^{\text{long}}) = r \cdot arccos(sin(u_1^{\text{lat}}) \cdot sin(u_2^{\text{lat}}) + cos(u_1^{\text{long}} - u_2^{\text{long}}))$$

2.3 Social Similarity

There are two main factors that contribute to someone's importance in a social graph: There are on the one hand nodes that have a lot of connections (sometimes referred to as "hubs" or "influencers") - on Twitter this may be Justin Bieber or Barack Obama. On the other hand you have nodes that are close to the query node - these people may be considered important because they are in close relation to the query. Please note that the first set of nodes is query-independent: Justin Bieber will always have the same amount of followers regardless of what node you are looking at. So both measures of importance for the query have to be taken into account. By how much is defined by a personalisation factor α. α may be determined by empirical studies and may be application-dependant or even query-dependant and chosen by the user, so we are considering α as variable and do not suggest or assume any specific value for it, besides it having to be from the range of $[0, 1]$.

Random Walk with Restart To measure similarity between nodes, a widely adopted model is Random Walk with Restart (RWR). It correlates with how close a node A is to a query node Q by considering not only direct edges or shortest distance (network distance), but also taking into account the amount of paths that exist between Q and A. With RWR, a virtual random walker starts at Q and then chooses any outgoing link by random. The walker will continue to do this, but with every iteration there is

a probability of α of jumping back to the query node (restart). The similarity between A and Q then is the probability of the random walker reaching A when starting at Q. Thus, the walker will visit nodes "close" to Q more likely than others, giving them a higher similarity. With the walker not being dependant on a single route between A and Q, modification of the nodes and links in the graph in general affects the similarity of all the other nodes through the addition or removal of possible paths the walker may choose from. Therefore, precomputing social similarities will become difficult and unpractical: In popular social networks links and nodes are added constantly, making precomputed similarities superfluous.

Bookmark Coloring Algorithm The Bookmark Coloring Algorithm (BCA) introduced by [4] is mathematically equivalent to RWR, but is more tangible and has other advantages. In a nutshell, it starts by injecting an amount of color into Q. Every node has the same color retainment coefficient α; i.e. that for every amount of color c the node receives, it gets to keep $\alpha \cdot c$, while the rest is forwarded equally spread across all outgoing links. This is the same α as for RWR. While BCA as well as RWR both rely on potentially infinite iterations of a power method to get exact results, one can terminate early to get quite exact approximations of them. But BCA follows more of a breadth-first-approach, while RWR may be compared to a depth-first one. This results in BCA giving all the socially close nodes a first visit much faster than RWR. We will exploit this feature to improve the search for socially close nodes over distant ones. It terminates when a stopping criterion is met; usually if the distributed color falls below a certain threshold or a total minimum sum of color is distributed). The algorithm gives back a vector, containing for every node its similarity to Q.

To derive the required social distance attribute for the skyline, we simply subtract the similarity, which lies within the bounds of $[0, 1]$, from 1. This fulfils the requirement that a node A with a similarity higher than that of node B (and is therefore considered better than B) gets a smaller distance than B (so that it is still better than B).

3 Related Work

Similarity Measures in Social-Networks: While colloquially speaking of "socially close" people, it is necessary to define a method of measuring this proximity. For this purpose, [15] proposed to use Random Walks with Restart ([24]) to measure social proximity. This metric, which is commonly adapted by the research community ([3, 9, 25]) to measure social similarity, considers all walks between two users, rather than using shortest path distance only ([18]), or using direct friendship relations of a user only [3].

Geo-Social Networks: [3] formulates a framework for geo-social query processing that builds queries based upon atomic operations. These queries (like *Nearest Friends*, *Range Friends*) focus on a specific user and his direct friends. [1] focusses on proximity detection of friends while preserving privacy through the fact that the location of a user is only known to the user himself and his friends, thus also lacking transitivity, where friends-of-friends are considered as well. [3] gives a comprehensive overview of the state-of-the-art in geo-social networks and geo-social queries.

Empirical Analysis: There are studies that examine the data of geo-social networks empirically: [7] detects that one's friends have a high probability of living close to each other, which emphasizes the importance of our introduced skyline queries when going further away from home. Similar findings have been made for the Foursquare geo-social network [22].

Skyline Queries: The skyline operator was introduced in [5]. Additionally, the authors propose block-nested-loop processing and an extended divide-and-conquer approach to process results for their new method. Since then, skyline processing has attracted considerable attention in the database community. [23] proposes two progressive methods to improve the original solutions. The first technique employs Bitmaps and is directed towards data sets being described with low cardinality domains. In other words, each optimization criterion is described by a small set of discrete attribute values. Other solutions for this scenario are proposed in [19]. The second technique proposed in [23] is known as index method and divides the data set into d sorted lists for d optimization criteria. [16] introduces the nearest-neighbor approach which is based on an R-Tree [11]. This approach starts with finding the nearest neighbor of the query point which has to be part of the skyline. Thus, objects being dominated by the nearest neighbor can be pruned. Afterwards, the algorithm recursively processes the remaining sections of the data space and proceeds in a similar way. A problem of this approach is that these remaining sections might overlap and thus, the result has to be kept consistent. To improve this approach, [20] proposes a branch-and-bound approach (BBS) which is guaranteed to visit each page of the underlying R-Tree at most once. There exist several techniques for post-processing the result of skyline queries for the case that the number of skyline points becomes too large to be manually explored (i.e. [21, 17]). Though we do not focus on reducing the number of results of our algorithm we can utilize the techniques of the above works in a post-processing step. There has already been some work considering the application of the skyline operator in a setting including road networks (i.e. [8, 13, 14]). The main difference to our setting is that the network distance in terms of the length of the shortest path is taken into consideration. Rather, in our work we use the RWR distance to compute the score among the individuals in the network graph which we will show poses new problems and challenges.

4 Social Distance Approximation

The exact evaluation of the social distance between two users using the RWR distance is computationally very expensive. Thus we rely on the following lower and upper bounds for the social similarity in order to boost the efficiency of our approach by avoiding having to calculate the social similarity to a precision not necessary for skyline queries, while still maintaining correctness of the results.

4.1 Bounds derived from network distance

Starting from the BCA algorithm, consider the maximum amount of color a node can get: To give a node A as much color as possible, the color has to flow directly from Q to A on the shortest path (that is the path with the least hops). With every hop, an α-portion

of the available color is "lost", because it gets assigned to the intermediate node. If the flow is not on the shortest path, at least another hop is added where another α-portion is lost and cannot contribute to A. Assume the shortest path from Q to A contains l edges. Then there are $l-1$ intermediate nodes plus one start node (Q), each of which gets its α-portion. The maximum amount of color that then can reach A in the best case is $(1-\alpha)^l$. This is an upper bound and can be computed for every node before even starting the BCA. The bound gets lower and "better", if either A has a larger distance to Q or α is large (high personalization).

In practice, calculating the network distance at query time (online phase) is not suitable. Dijkstra's algorithm for example gets expensive for a large network with many nodes. Therefore, we suggest to introduce a preprocessing step that supports approximating the network distance optimistically. We use graph embedding [10], where the distance from any node to a small set of reference nodes *RN* is precomputed and then stored in a lookup-table. At query-time, we can then easily derive a conservative as well as an optimistic approximation for the network distance. Let o be the optimistic approximation (that is $o \leq$ actual network distance l). Then

$$(1-\alpha)^o \geq (1-\alpha)^l \geq \mathit{BCA}(Q)_A$$

where $\mathit{BCA}(Q)_A$ is the actual amount of color node A gets assigned. Therefore, the approximation derived from precomputed graph embedding provides an upper bound as well.

4.2 Bounds derived from BCA

In general, it is possible that a node does not receive any color at all, so its generic lower bound is 0. But if the node has already received some color, there is no way of it ever losing this amount of assigned color - so once assigned color can be interpreted as an ever-improving lower bound. Building upon this, another upper bound can be derived from this lower bound: The maximum amount of color this node can get is the color it already has plus the total remaining unassigned color in the BCA-queue.

5 Algorithm

In the following, we propose three algorithms to compute the Geo-Social Skyline: First we provide a straightforward solution which is simple but not practicable for large datasets. Then, we propose a baseline solution which utilizes the bounds proposed in Section 4 and an advanced solution which uses additional pruning criteria to further improve its performance.

5.1 Naive Algorithm

Assuming no index structure, neither for the social nor for the spatial dataset, a naive algorithm for computing the Geo-Social Skyline is shown in Algorithm 1. This approach computes spatial distances and social distances of all users in Line 4 and Line 5 respectively. Computation of social distance requires to call the complete Bookmark Coloring

Algorithm 1 Naive Geo-Social Skyline Computation

Require: SN, q_{geo}, q_{soc}, α
1: bca = *PerformCompleteBCA*(SN, q_{soc}, α)
2: **for** each $n \in SN$ **do**
3: **if** $n_{\text{soc}} \neq q_{\text{soc}}$ **then**
4: n.spatialDistance $= distance(n_{\text{geo}}, q_{\text{geo}})$
5: n.socialDistance $= 1 - bca_{n.\text{id}}$
6: **end if**
7: **end for**
8: compute *skyline* using a scan based skyline algorithm
9: **return** *skyline*

Algorithm in Line 1, which is the main bottleneck of this approach. Given spatial and social distances, we represent each user by a two-dimensional vector and apply traditional approaches to compute two-dimensional skylines in Line 8. This naive algorithm however is very ineffiecient and not practicable in a setting with large datasets. The main problem here is that the runtime complexity of BCA has shown to be $O(n^3)$. Thus the following algorithm makes use of the social distance bounds proposed in Section 4 in order to avoid the complete run of BCA.

5.2 Baseline Algorithm

The baseline algorithm assumes a simple index structure such as a sorted list, a min-heap or a B-Tree to access users in ascending order to their spatial distance to the query location q_{geo}. This algorithm starts by computing spatial distance and initializing distance bounds in Lines 3-10. In Line 8, the lower bound is initialized by network distance bounds described in Section 4.1. The upper bound is initiliazed with the trivial upper bound of 1. Then, nodes are accessed in increasing order of the distance to q_{geo} in Line 12. When a new node c is accessed, a check is performed in Line 14 to see if c's lower bound social distance is already higher than the upper bound social of the last object that has been returned as a skyline result. In this case, c can be pruned, as we can guarantee that the previous result node *previous* has a lower social distance than c and due to accessing nodes ordered by their spatial distance, we can guarantee that *previous* must also have a lower spatial distance. Another check is performed in Line 16, for the case where c can be returned as a true hit, by assessing that c must have a lower social distance than *previous*. If neither of these two checks allows to make any decision, Line 19 calls Algorithm 3 to perform an additional iteration of the BCA algorithm to further refine all current social bounds, until c can either be pruned or returned as a true hit. The main idea of this algorithm is to minimize expensive iterations of the BCA. This is achieved by first considering the spatial dimension. Furthermore, BCA iterations are required only in cases where absolutely necessary. For this reason, the social distance of results may still be an approximation. Nevertheless, this algorithm can guarantee to return the correct skyline.

Note that in the worst case, where every node is contained in the skyline, this approach yields no performance gain compared to the trivial solution. A major disadvan-

Algorithm 2 Geo-Social Skyline Baseline

Require: q_{geo}, q_{soc}, α, SN, RN
1: $assigned = [], incoming = [], total = 0, results = \emptyset$
2: $queue = \emptyset$ // min-heap sorted ascending by $dist_{\text{geo}}$
3: **for** $n \in SN$ **do**
4: $incoming[n.\text{id}] = 0, assigned[n.\text{id}] = 0$
5: **if** $n \neq q_{\text{soc}}$ **then**
6: $n.\text{spatialDistance} = dist_{\text{geo}}(n.\text{point}, q_{\text{geo}})$
7: $l = max_{R_i \in RN}(|networkDist(q_{\text{soc}}, R_i) - networkDist(n, R_i)|)$
// those network distances come from graph embedding
8: $n.\text{socialDist.lower} = 1 - (1-\alpha)^l, n.\text{socialDist.upper} = 1, queue.add(n)$
9: **end if**
10: **end for**
11: $incoming[q_{\text{soc}}.\text{id}] = 1, previous = queue.popMin(), results.add(previous)$
12: **while** $queue.\text{size} > 0$ **do**
13: $candidate = queue.popMin()$
14: **if** $previous.\text{socialDist.upper} < candidate.\text{socialDist.lower}$ **then**
15: $prune(candidate)$
16: **else if** $candidate.\text{socialDist.upper} < previous.\text{socialDist.lower}$ **then**
17: $results.add(candidate), previous = candidate$
18: **else**
19: $IncrementalBCA(\alpha, \epsilon, incoming, assigned, total)$ // see Algorithm 3
20: $queue.add(candidate)$ // re-insert
21: **end if**
22: **end while**
23: **return** *results*

tage of this approach is its space consumption and the linear scan over the database to materialize every single node, for which the spatial distances are calculated and the sorting takes place. This means that basically the entire graph has to be loaded into memory for processing. The following algorithm alleviates this problem.

5.3 Improved Algorithm

Our improved algorithm still iterates over nodes from closest to farthest spatially, but avoids having to store the whole graph in memory. Therefore, we use a spatial index structure such as an R-Tree, which supports efficient incremental nearest neighbour algorithms [12] to access nodes sorted by increases distance to the query location q_{geo}. This approach allows to avoid loading spatial locations of nodes into the memory if we can terminate the algorithm early by identifying a time when we can guarantee that all skyline points have been found.

For this purpose, the improved algorithm uses the bounds presented in Section 4.2 by considering the maximum amount of color these nodes can get in future iterations of the BCA. These bounds essentially allow to assess an upper bound of completely unseen, i.e., not yet accessed nodes. Given these bounds, and exploiting that unseen nodes must have a higher spatial distance than all accessed nodes due to accessing

Algorithm 3 IncrementalBCA

Require: α, ϵ, *incoming*, *assigned*, *total*

1: $color = \max(incoming[]), k = incoming.indexOf(color)$
2: $incoming[k] = 0, assigned[k] = assigned[k] + \alpha \cdot color$
3: $total = total + \alpha \cdot color$
4: **for** each $l \in k.getLinks()$ **do**
5: $\quad incoming[l] = incoming[l] + (1 - \alpha) \cdot color / k.getLinks().\text{size}$
6: **end for**
7: $k.suggestLB(total - assigned[k]), k.suggestUB(1 - assigned[k])$
8: **if** $k.\text{UB} - k.\text{LB} < \epsilon$ **then**
9: $\quad k.\text{LB} = k.\text{UB} = (k.\text{LB} + k.\text{UB})/2$
10: **end if**

nodes in ascending order of their spatial distance, we terminate the algorithm early in Line 3 if the following conditions are met:

- The set of candidates contains no entries anymore (i.e. all candidates either became results or were pruned) and
- all unseen nodes can be pruned, thus there is no possibility an unseen node can be contained in the skyline.

6 Experiments

For our experiments, we evaluated our solutions on a geo-social network taken from the Gowalla dataset[3]. It consists of a social network having 196,591 nodes and 950,327 undirected edges, leaving every node on average with approximately ten links. Furthermore, the dataset contains 6,442,890 check-ins of users. Each user is assigned their latest (most recent) check-in location as geo-spatial location. Users having no check-in at all are matched to a special location that has a geo-distance of infinity to any other place. For all geo-social skyline queries performed in this experimental evaluation, if not mentioned otherwise, the spatial query components q_{geo} are obtained by uniformly sampling $(latitude, longitute)$ pairs in the range $([-90, 90], [-180, 180])$. The social query components q_{soc} are obtained by uniformly sampling nodes of the social network.

When compared to trending social networks like Facebook or Foursquare, the example dataset is rather small when compared in network size. Unfortunately, data of larger networks is not publicly available. We decided not to run experiments on synthetic data, because the artificial generation of spatial and social data may introduce a bias into the experiments, when correlations between spatial and social distances are different in real and synthetic data. Although there exist indiviual data generators for social-only networks (e.g. [2]) and spatial networks ([6]), the naive combination of both generators is not feasible here.

In Figure 2 we compare all proposed algorithms with a varying value of α. The improved algorithm allows further optional domination checks in lines 12 and 13, which were not done for the *prune unseen* data row. Only the check in Line 12 is done in *prune*

[3] http://snap.stanford.edu/data/loc-gowalla.html

Algorithm 4 Geo-Social Skyline Improved

Require: q_{geo}, q_{soc}, α, *SN*, *RN*

1: $assigned = [], incoming = [], total = 0, results = \emptyset, prunedUnseen =$ **false**
2: $previous = getNearestNeighbour(q_{\text{geo}}), results.add(previous), i = 2, incoming[q_{\text{soc}}.\text{id}] = 1$
3: **while** $i < |SN.\text{Nodes}|$ **and not** *prunedUnseen* **do**
4: $candidate = getN^{th}NearestNeighbour(q_{\text{geo}}, i)$ // get i^{th} nearest neighbour of q_{geo}
5: i++
6: **if not** *candidate*.isPruned **then**
7: **while** $[candidate.\text{LB}, candidate.\text{UB}] \cap [previous.\text{LB}, previous.\text{UB}] \neq \emptyset$ **do**
8: $IncrementalBCA(\alpha, \epsilon, incoming, assigned, total)$ // see Algorithm 3
9: **end while**
10: **if not** $previous.dominates(candidate)$ **then**
11: $results.add(candidate)$
12: // optional: check if candidate dominates other candidates (remove them)
13: // optional: check if elements in candidates dominate each other (remove them)
14: $previous = candidate, prunedUnseen = (candidate.\text{UB} < total)$
15: **else**
16: $prune(candidate)$
17: **end if**
18: **end if**
19: **end while**
20: **return** *results*

candidate with hit, while both checks are done in the *prune candidates with candidates* rows. For the latter, the frequency of the check in Line 13 is varied for every 1,000 respectively 10,000 iterations.

The experiments show that both optional checks in Algorithm 4 do not result in better runtimes with our datasets, even when only performed at greater intervals (that is not after every single iteration of BCA). On the other hand, checking whether all unseen nodes can be pruned results in an actual performance increase compared with the baseline algorithm. As a result of this experiment, all following experiments will use the *prune unseen* setting, as this algorithm shows the best runtime performance for almost any $\alpha \in [0, 1]$.

6.1 Skyline Results

For varying α, we experience a rapid drop in runtime for a larger α as seen in Figure 3(a). This can be attributed to the fact that a larger value of α allows the BCA to terminate faster, as in each iteration more color is distributed over the social network, thus yielding tighter bounds in each iteration. This result implies that, since α represents the personalization factor of the query, highly personalized queries can be performed much faster than unpersonalized ones. This is evident, since unpersonalized queries represent an overview of the entire network, thus requiring a deeper and more complete scan of the graph. Choosing an α-value of zero will cause the BCA not terminating, because no color gets assigned. An α-value approaching zero causes the BCA to approach the stationary distribution of the social network, which is completely independent of the

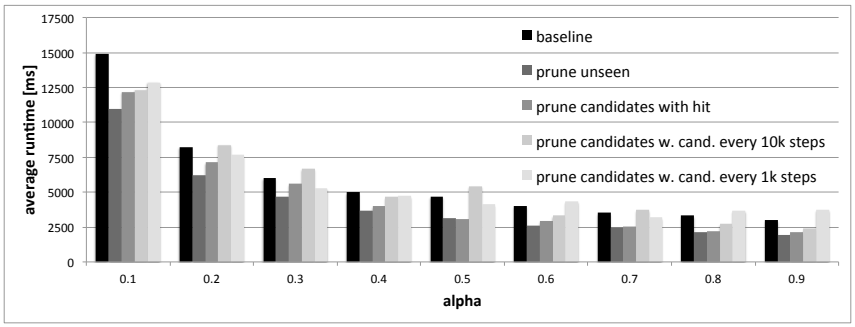
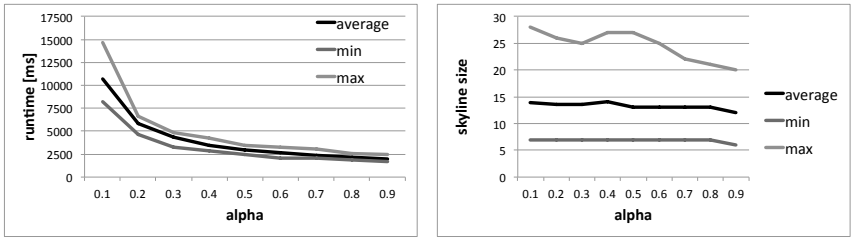


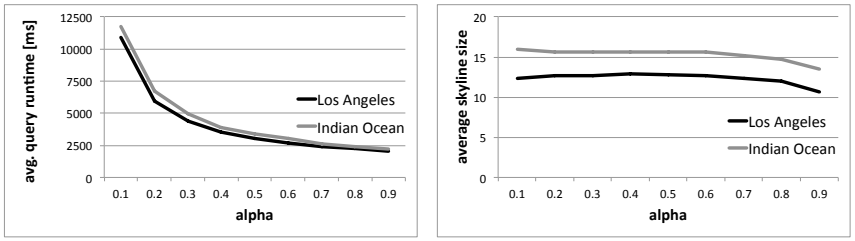
Fig. 2. Runtime Evaluation



(a) Runtime (b) Skyline Size

Fig. 3. Evaluation of α.

social query node q_{soc}. A value of α approaching one will give all the colour to the q_{soc}, thus returing q_{soc} himself as his only match. Furthermore, it can be observed in Figure 3(b) that despite a fairly large social network, the number of results in the Geo-Social Skylines become quite small. We ran several queries for different α-values and in our experiments the number of elements in the skyline remained low - we never encountered a skyline containing more than approximately 30 elements. This shows that the spatial-social skyline returns useful result sets for real-world applications by reducing a large network of approximately 200k nodes down to a feasible number of elements.

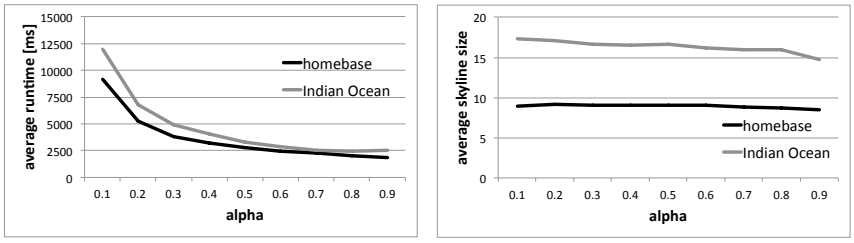


(a) Runtime (b) Skyline Size

Fig. 4. Evaluation of the population density of q_{geo}.

6.2 Varying the Spatial Query Point

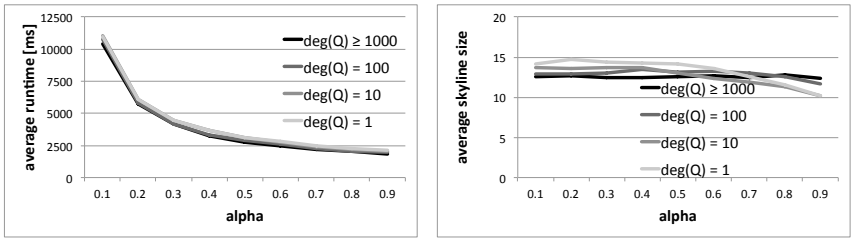
Considering the motivational example of querying the graph when going on vacation, it becomes interesting to query areas considered "dense" and "sparse" populated places. In our setting, we considered regions to be dense where a lot of social nodes are located at. In the Gowalla data set, the city of Los Angeles is such a dense place, as the user base of this geo-social network was mainly U.S.-based. In contrast a place in the southern Indian Ocean is chosen as a sparse region. This region is located approximately on the opposite of the U.S. on the Earth, so this region maximizes the spatial distance of the majority of the user base. In the following experiment, we performed pairs of geo-social skyline queries, such that each pair had an identical social query user q_{soc}, but the spatial location q_{geo} differes by being either in Los Angeles and in the Indian Ocean. While we only observed a slight performance gain when querying dense locations (cf. Figure 4(a)), it is interesting to see in Figure 4(b) that querying a sparse place results in a larger skyline. The reason is that users in the U.S. have a higher average number of social links in this data set than users outside of the U.S., since most active users of this geo-social network origin from the U.S. It is more likely for a random user to have a socially close person coming from the vicinity of Los Angeles, therefore having both a small social and a small geographic distance from a query issued in Los Angeles, and thus pruning most of the database.



(a) Runtime (b) Skyline Size

Fig. 5. Evaluation of the distance between the user and q_{geo}.

In another set of experiments we evaluate the effect of the spatial distance between a user that performs a geo-social skyline query and the spatial query location q_{geo}. Therefore, we performed pairs of geo-social skyline queries, such that each pair had an identical social query user q_{soc}, but the spatial location q_{geo} differed by either being identical to the user's location, and by being located in the Indian Ocean. The result depicted in Figure 5(b) shows that the skyline for the home-based query only contains roughly half as many elements as the distant one also yielding a lower runtime (cf. Figure 5(a)). The reason is that in geo-social networks, the spatial distance between users is known to be positively correlated with their social distance [22]. Thus, it is likely to have a socially close person in your immediate vicinity - and in the case where a user's location equals the query point q_{geo}, this person dominates most other individuals. At the same time, the other extreme case where q_{geo} is located at the other end of the world, leads to a negative correlation between social distance and distance to q_{geo} thus yielding a larger number of skyline results.



(a) Runtime (b) Skyline Size

Fig. 6. Evaluation of the number of friends of q_{soc}.

6.3 Varying the Number of Friends of q_{soc}

Next, we evaluated in impact of the degree of q_{soc}, i.e., the number of direct friends of the query user, as we expect that social stars may produce skylines shaped differently than users having a few friends only. The experimental results depicted in Figure 6 show that, while making almost no difference in terms of run-time, we can observe, for small values of α, a slight trend towards larger skylines for nodes having a low degree. For large values of α, this effect reverses.

7 Conclusions

In this work, we have defined the problem of answering *Geo-Social Skyline Queries*, a useful new type of query that allows to find, for a specified query user and a specified spatial query location, a set of other users that are both socially close to the query user and spatially close to the query location. To answer such queries, we followed the state-of-the-art approach of measuring social distance using Random-Walk-with-Restart-distance (RWR-distance), which is more meaningful than simple social distance measures such as the binary *isFriend*-distance and the shortest-path distance. Due to the computational complexity of computing RWR-distances, we have presented efficient techniques to obtain conservative bounds of RWR-distances which can be used to quickly prune the search space, thus alleviating computational cost significantly as shown by our experimental studies.

As a future step, we want to exploit information given by check-in data provided by users, rather than using their current spatial position only. Such check-in data allows to automatically return for a given a user, who is going to visit a location such as the island of Bali, friends that have recently visited Bali. These friends can be recommended to the user as experts that may help the user to, for example, find good places to visit.

References

1. A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler. Buddy tracking - efficient proximity detection among mobile friends. *Pervasive and Mobile Computing*, 3(5):489–511, 2007.

2. R. Angles, A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. Benchmarking database systems for social network applications. In *Proc. GRADES*. ACM, 2013.
3. N. Armenatzoglou, S. Papadopoulos, and D. Papadias. A general framework for geo-social query processing. In *Proc. VLDB*, pages 913–924. VLDB Endowment, 2013.
4. P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006.
5. S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430. IEEE, 2001.
6. T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
7. E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proc. KDD*, pages 1082–1090. ACM, 2011.
8. K. Deng, Y. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *Proc. ICDE*, pages 796–805. IEEE, 2007.
9. Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for random walk with restart. In *Proc. VLDB*, pages 442–453. VLDB Endowment, 2012.
10. F. Graf, H.-P. Kriegel, M. Renz, and M. Schubert. Memory-efficient A*-search using sparse embeddings. In *Proc. ACM GIS*, pages 462–465. ACM, 2010.
11. A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, pages 47–57. ACM, 1984.
12. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.
13. X. Huang and C. S. Jensen. In-route skyline querying for location-based services. In *Proc. W2GIS*, pages 120–135. Springer, 2004.
14. S. M. Jang and J. S. Yoo. Processing continuous skyline queries in road networks. In *Proc. CSA*, pages 353–356. IEEE, 2008.
15. I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *Proc. SIGIR*, pages 195–202. ACM, 2009.
16. D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *Proc. VLDB*, pages 275–286. VLDB Endowment, 2002.
17. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: the k most representitive skyline operator. In *Proc. ICDE*, pages 86–95. IEEE, 2007.
18. W. Liu, W. Sun, C. Chen, Y. Huang, Y. Jing, and K. Chen. Circle of friend query in geo-social networks. In *Proc. DASFAA*, pages 126–137. Springer, 2012.
19. M. Morse, J. M. Patel, and H. Jagadish. Efficient skyline computation over low-cardinality domains. In *Proc. VLDB*, pages 267–278. VLDB Endowment, 2007.
20. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. SIGMOD*, pages 467–478. ACM, 2003.
21. J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: a semantic approach based on decisive subspaces. In *Proc. VLDB*, pages 253–264. VLDB Endowment, 2005.
22. S. Scellato, C. Mascolo, M. Musolesi, and V. Latora. Distance matters: geo-social metrics for online social networks. In *Proc. WOSN*. USENIX, 2010.
23. K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proc. VLDB*, pages 301–310. VLDB Endowment, 2001.
24. H. Tong, C. Faloutsos, and J. Y. Pan. Fast random walk with restart and its applications. In *Proc. ICDM*, pages 613–622. IEEE, 2006.
25. C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. Evaluating geo-social influence in location-based social networks. In *Proc. CIKM*, pages 1442–1451. ACM, 2012.