Geo-Social Keyword Skyline Queries

Naoya Taguchi[1(✉)], Daichi Amagata[2], and Takahiro Hara[2]

[1] University of Tokyo, Tokyo, Japan
guchio@logos.t.u-tokyo.ac.jp
[2] Osaka University, Suita, Japan
{amagata.daichi,hara}@ist.osaka-u.ac.jp

Abstract. Location-Based Social Networking Services (LBSNSs) have been becoming increasingly popular. One of the applications provided by LBSNSs is a PoI search based on spatial distance, social relationships, and keywords. In this paper, we propose a novel query, Geo-Social Keyword Skyline Query (GSKSQ), which returns the skyline of a set of PoIs based on a query point, the social relationships of the query owner, and query keywords. Skyline is the set of data objects which are not dominated by others. We also propose an index structure, Social Keyword R-tree, which supports efficient GSKSQ processing. The results of our experiments on two real datasets Gowalla and Brightkite demonstrate the efficiency of our solution.

1 Introduction

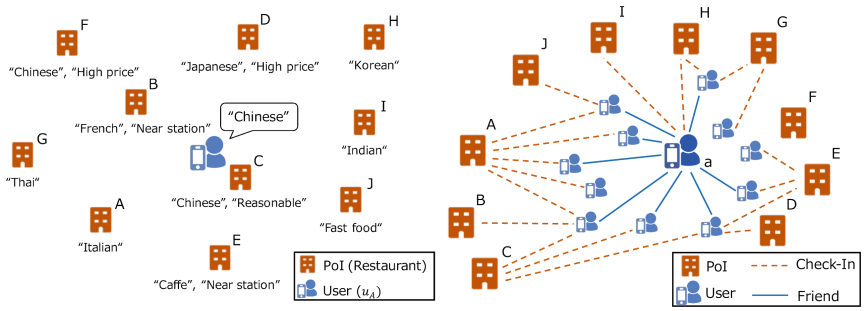
With the wide spread of mobile devices such as smart phones and tablets, people use Location-Based Services (LBS) and Social Networking Services (SNS) in their daily lives [4,8]. Due to this fact, Location-Based Social Networking Services (LBSNS), e.g., Facebook and Yelp, are also prevalent [3]. In a LBSNS, we can search points of interests (PoIs) based on three criteria, spatial distance, social relationships, and keywords [1].

In this paper, we propose a novel query, Geo-Social Keyword Skyline Query (GSKSQ), which returns the skyline [2] of a set of PoIs based on a query point, the social relationships of the query owner, and user-specified keywords. Skyline is the set of data objects which are not dominated by others. Informally, a data object o is dominated by another data object o' if o is worse than o' for all attributes. The result of a GSKSQ thus supports multi-criteria decision making in LBSNS applications. We show a practical example of a GSKSQ in Example 1.

Example 1. *Assume that a user u_a searches a restaurant on a LBSNS, and u_a can choose it based on his/her current position, the number of check-ins of his/her friends, and the number of keywords matched with a query. In this case, u_a can find such a restaurant easily by using a GSKSQ. Figure 1a represents an example situation where u_a specifies his/her position as a query point and "Chinese restaurant" as a query keyword. Figure 1b represents the corresponding social graph. As we can see in Figs. 1a and b, restaurant B is worse than restaurant C in the three attributes, so B is dominated by C. In the same manner,*

D. Benslimane et al. (Eds.): DEXA 2017, Part I, LNCS 10438, pp. 425–435, 2017.
DOI: 10.1007/978-3-319-64468-4_32

we can see that restaurants D, E, F, G, H, I, and J are dominated by C. As a result, u_a obtains restaurants A and C as the query result, and he/she can easily choose preferable one from these two restaurants.



(a) An example of user-specified keyword (Chinese) and a set of restaurants

(b) A social graph example

Fig. 1. A practical example of a GSKSQ

A naive way to calculate a Geo-Social Keyword Skyline (GSKS) is to check whether or not a given PoI is dominated by any other PoIs. However, this approach is computationally expensive particularly when the number of PoIs is large [2]. In addition, because the result of a GSKSQ is dependent on a query, we cannot pre-compute the results of any queries. In this paper, we propose an index structure called Social Keyword R-tree (SKR-tree), which is a kind of aR-tree [5], and supports efficient GSKSQ processing. With the SKR-tree, we can retrieve the result while pruning unpromising PoIs. Our contributions in this paper are summarized as follows: (1) We propose a novel query, GSKSQ, which returns the skyline of a set of PoIs based on a query point, the social relationships of the query owner, and user-specified keywords. (2) We propose an index structure called SKR-tree to calculate a GSKS efficiently. (3) We propose a scoring function to support efficient retrieval on SKR-tree. We design this function based on an idea about user behaviors, and show that this function is optimal w.r.t. the number of node accesses of the SKR-tree. (4) The results of our experiments on real datasets Gowalla[1] and Brightlite[2] demonstrate the efficiency of our solution.

The organization of this paper is as follows. Section 2 defines the problem of this paper. We describe our algorithm in Sect. 3, and Sect. 4 presents our experimental results. Finally, we conclude this paper in Sect. 5.

[1] https://snap.stanford.edu/data/loc-gowalla.html.
[2] https://snap.stanford.edu/data/loc-brightkite.html.

2 Problem Definition

2.1 Geo-Social Keyword Skyline Query (GSKSQ)

We define a PoI $p_i \in P$ as $p_i = \langle loc, key \rangle$, where i is the identifier of p_i, loc is the location of p_i, and key is the set of keywords held by p_i. When a user u retrieves a GSKS, he/she issues a GSKSQ by specifying a query point and query keywords. A GSKSQ is defined as $q_u = \langle loc, key \rangle$. In response to q_u, each $p_i \in P$ obtains attribute values $p_i.G(q_u)$, $p_i.S(q_u)$, and $p_i.K(q_u)$, which respectively correspond to a spatial distance, social relationships, and keywords. $p_i.G(q_u)$ is the Euclidean distance between $q_u.loc$ and $p_i.loc$, which is defined as follows.

$$p_i.G(q_u) = dist(q_u.loc, p_i.loc)$$

Note that smaller $p_i.G(q_u)$ is better. $p_i.S(q_u)$ is the number of users who are friends of u and have checked-in to p_i. So, $p_i.S(q_u)$ is defined as

$$p_i.S(q_u) = |u.friends \cap p_i.checkin|,$$

where $u.friends$ is the set of users who are friends of u, and $p_i.checkin$ is the set of users who have checked-in to p_i. For example, in Fig. 1b, $A.S(q_u)$ is 4, $C.S(q_u)$ is 3, and $F.S(q_u)$ and $I.S(q_u)$ are 0. Larger $p_i.S(q_u)$ is better. $p_i.K(q_u)$ is the number of the common keywords in $q_u.key$ and $p_i.key$, and is defined as follows.

$$p_i.K(q_u) = |q_u.key \cap p_i.key|$$

As same as $p_i.S(q_u)$, larger $p_i.K(q_u)$ is better. Based on these three attribute values, we define dominance below.

Definition 1 (Dominance). *Let q_u be a geo-social keyword skyline query. If p_i and p_j satisfy that $(p_i.G(q_u) \geq p_j.G(q_u)) \wedge (p_i.K(q_u) \leq p_j.K(q_u)) \wedge (p_i.S(q_u) \leq p_j.S(q_u))$, we represent this condition as follows.*

$$p_i \preceq p_j$$

Furthermore, if p_i and p_j satisfy that $(p_i \preceq p_j) \wedge [((p_i.G(q_u) > p_j.G(q_u)) \vee (p_i.K(q_u) < p_j.K(q_u)) \vee (p_i.S(q_u) < p_j.S(q_u))]$, p_i is dominated by p_j, and we represent this condition as follows.

$$p_i \prec p_j$$

We now define a GSKS as follows.

Definition 2 (Geo-Social Keyword Skyline (GSKS)). *Let P be a set of PoIs, and $q_u = \langle loc, key \rangle$ be a geo-social keyword skyline query. The Geo-Social Keyword Skyline of P calculated on q_u is the subset P' of P, which satisfies that*

$$\nexists p_j \in P \text{ such that } p_i \prec p_j, \tag{1}$$

for $\forall p_i \in P'$.

Our objective is to achieve efficient processing of a GSKSQ.

Algorithm 1. Baseline algorithm

```
1  for ∀p_i ∈ P do
2      Calculate each attribute value of p_i
3  for ∀p_i ∈ P do
4      for ∀p_j ∈ P do
5          if p_i ≻ p_j then
6              P ← P\{p_j}
7          if p_i ≺ p_j then
8              P ← P\{p_i}
9              break
10 return P
```

2.2 Baseline

We show a baseline algorithm for processing a GSKSQ in Algorithm 1. In Algorithm 1, we calculate all attribute values of all PoIs at first (lines 1–2). We then check whether or not a given PoI dominates or is dominated by other PoIs (lines 3–9), and finally we obtain a GSKS (line 10).

Algorithm 1 is computationally expensive because of two reasons. First, we have to calculate all attribute values of all PoIs. In particular, it increases computational cost w.r.t. $p_i.S(q_u)$. This is because to calculate this attribute value, we have to check how many friends of the query owner checked-in for all PoIs. Second, the baseline algorithm executes dominance check for all PoIs, and this operation is the main overhead of skyline computation [7]. We propose a more efficient algorithm which alleviates this cost.

3 Proposed Solution

3.1 Social Keyword R-Tree (SKR-Tree)

We propose an index structure called SKR-Tree to achieve efficient GSKSQ processing. SKR-tree is a kind of a R-tree as illustrated in Fig. 2. Nodes of SKR-tree store information on the three attributes, and are classified to two kinds of nodes, leaf nodes and internal nodes. A leaf node n_i corresponds to a PoI p_i, and contains:

- a pointer to p_i,
- $n_i.loc$, which is the location of p_i,
- $n_i.key$, which is the set of keywords held by p_i,
- $n_i.S$, which is the upper-bound social value of p_i.

We detail $n_i.S$, which is the upper bound of $p_i.S(q_u)$ for any q_u, in Sect. 3.2. For example, in Fig. 2, leaf node n_A (n_B) contains a pointer to p_A (p_B), $n_A.key = \{k_1\}$ ($n_B.key = \{k_2\}$) is the set of keywords, and $n_A.S = 1$ ($n_B.S = 3$) is the

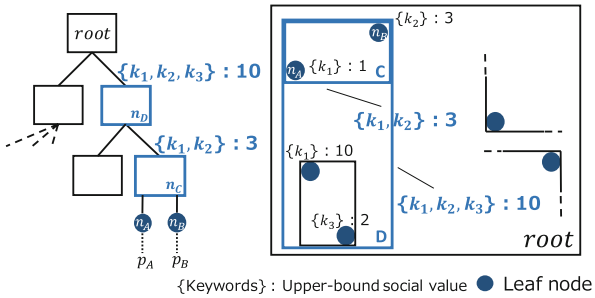


Fig. 2. An example of SKR-tree

upper-bound social value. An internal node n_i is a Minimum Bounding Rectangle (MBR) which consists of its all child nodes, and contains:

- $n_i.key$, which is the union of the keyword sets which are held by its all child nodes,
- $n_i.S$, which is the maximum upper-bound social value among all its child nodes' ones.

For example, in Fig. 2, internal node n_C is an MBR which consists of its child nodes n_A and n_B, a set of keywords $\{k_1, k_2\} = n_A.key \cap n_B.key$, and 3, which is the maximum of $n_A.S$ and $n_B.S$.

Given a GSKSQ q_u, we can calculate the three attribute values for all n_i, and we represent them as $n_i.G(q_u)$, $n_i.S$, and $n_i.K(q_u)$. If n_i is a leaf node, $n_i.G(q_u)$ is $p_i.G(q_u)$. If n_i is an internal node, $n_i.G(q_u)$ is the Euclidean distance between $q_u.loc$ and the nearest neighbor point in the corresponding MBR of n_i. $n_i.S$ is an upper-bound social value of n_i, and $n_i.K(q_u)$ is $|q_u.key \cap n_i.key|$. By using these attribute values, we can execute dominance check between a node n_i and a PoI p_j. If n_i is dominated by p_j, PoIs which are pointed by n_i's descendant nodes are not in the GSKS, so we can prune the subtree rooted at n_i. This reduces attribute value calculation, node accesses, and dominance checks. Therefore, we can accelerate query processing performance. Note that once SKR-tree is constructed, it is updated efficiently by incremental manner [5].

3.2 Upper-Bound Social Value Calculation

To construct SKR-tree, we have to calculate upper-bound social values, which are defined below, for all leaf nodes.

Definition 3 (Upper-bound social value). *Given a leaf node n_i and a set of all users U, $n_i.S$ is calculated as follows.*

$$n_i.S = max_{\forall u_j \in U} |u_j.friends \cap p_i.checkin|$$

For example, in Fig. 3, $|u_a.friends \cap p_A.checkin|$ is 3, and those for u_b, u_c, u_d, u_e, and u_f are respectively 3, 1, 1, 4, and 2. In this case, $n_A.S = 4$. Note that once we obtain $n_i.S$, it is efficiently updated if the social graph is updated because an insertion or a deletion of a check-in and a friendship affects just a part of social graph.

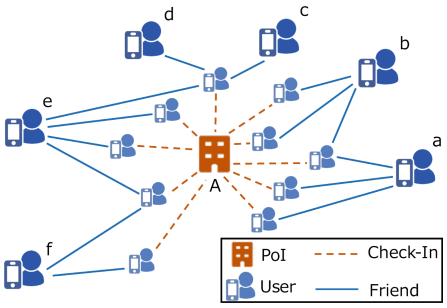


Fig. 3. An example social graph to calculate an upper-bound social value

3.3 GSKSQ Processing Algorithm

We propose Algorithm 2 to calculate a GSKS efficiently on a SKR-tree.

In Algorithm 2, we traverse nodes based on scores. These scores are obtained by a scoring function F, and higher scores mean higher priorities. We achieve this priority by using a priority queue Q_F, and this queue is initialized by the root node n_{root} (line 1). At the first of each iteration, we pop and get the top node of Q_F, n_{temp}, and check whether or not n_{temp} is dominated by PoIs in P^{temp}_{GSKS}, which is an intermediate result set (lines 3–8). If n_{temp} is dominated by $p_i \in P^{temp}_{GSKS}$, we prune the subtree rooted at n_{temp}, and proceed to the next iteration. Otherwise, if its child nodes $n_i \in n_{temp}.children$ are internal nodes, we calculate the three attribute values of n_i, and push them to Q_F (lines 9–13). If $n_i \in n_{temp}.children$ are leaf nodes, we calculate $p_i.G(q_u)$, $p_i.S(q_u)$, and $p_i.K(q_u)$ where p_i is pointed by n_i. Then, we execute dominance checks for all p_i against the PoIs in P^{temp}_{GSKS}. After that, we add the PoIs pointed by nodes in $n_{temp}.children$ which are not dominated by $p_j \in P^{temp}_{GSKS}$ to P^{temp}_{GSKS}, and remove non-skyline PoIs from P^{temp}_{GSKS} (lines 14–23). We continue this iteration while Q_F is not empty (line 24).

3.4 Scoring Function Design

The node traversal order in Algorithm 2 is dependent on a scoring function F. We design F based on two criteria. First one is the number of node accesses. To achieve fast query processing, the number of node accesses should be minimized.

Algorithm 2. PROPOSED ALGORITHM

```
1  Q_F.push(n_root)
2  while Q_F ≠ ∅ do
3      n_temp = Q_F.top
4      Q_F.pop
5      for ∀p_i ∈ P_sky^temp do
6          if n_temp ≺ p_i then
7              n_temp.children = ∅
8              break
9      if n_temp.children are not leaf then
10         for ∀n_i ∈ n_temp.children do
11             Calculate each attribute value of n_i
12             n_i.score = F(n_i, q_u)
13             Q_F.push(n_i)
14     else
15         for ∀n_i ∈ n_temp.children do
16             Calculate each attribute value of n_i.p
17             P_sky^temp ← P_sky^temp \ {∀p_j ∈ P_sky^temp | p_j ≺ n_i.p}
18             if ∄p_j ∈ P such that p_i ≺ p_j then
19                 P_sky^temp ← P_sky^temp ∪ {n_i.p}
20 return P_sky^temp
```

Second one is the number of dominance checks. It is intuitively known that if P_{GSKS}^{temp} has a skyline PoI that dominates many PoIs, we can reduce unnecessary checks. Therefore, we should obtain such a PoI as soon as possible.

Now we address the first requirement and introduce Theorem 1. Proof is omitted due to space limitation.

Theorem 1. *Assume that we process a GSKSQ by using Algorithm 2. Let n_i and n_j be arbitral nodes of SKR-tree, which satisfy that $n_i \prec n_j$. If F satisfies that*

$$F(q_u, n_i) < F(q_u, n_j), \tag{2}$$

the number of node accesses is minimum for retrieving the GSKS on the SKR-tree.

Based on Theorem 1, we address the second requirement. Our F consists of three sub-scoring functions f_G, f_S, and f_K, which correspond to the three attributes. As well as F, higher scores of these sub-scoring functions mean higher priorities. Note that, PoIs which dominate larger space potentially dominate may nodes and PoIs. Here, we design F and the three sub-scoring functions as follows.

$$F(q_u, n_i) = \begin{cases} f_G(q_u, n_i) \cdot f_S(q_u, n_i) \cdot f_K(q_u, n_i) & (f_G(q_u, n_i) \neq 0) \\ -\frac{1}{f_S(q_u, n_i) \cdot f_K(q_u, n_i)} & (f_G(q_u, n_i) = 0) \end{cases}, \tag{3}$$

$$f_G(q_u, n_i) = dist_{max} - n_i.G(q_u), \tag{4}$$

$$f_S(q_u, n_i) = \begin{cases} n_i.S(q_u) & (n_i.S(q_u) \neq 0) \\ \alpha & (n_i.S(q_u) = 0) \end{cases}, \tag{5}$$

$$f_K(q_u, n_i) = \begin{cases} n_i.K(q_u) & (n_i.K(q_u) \neq 0) \\ \alpha & (n_i.K(q_u) = 0) \end{cases}, \tag{6}$$

where $dist_{max}$ is the farthest distance among the ones between $q_u.loc$ and all corners of n_{root}, which is an MBR, and α is a real number which satisfies $0 < \alpha < 1$. The reason why F, f_S, and f_K have 2 cases is that the most natural definitions of them, which is the first cases, do not satisfy Eq. (2) in case that one or more of f_G, f_S, and f_K is 0.

In addition to this, we refine f_S by considering a practical characteristic of social relationships. Assume that a user issues a GSKSQ. It can be expected that he/she tends to issue this query in his/her friends' living area. This is because users on SNSs are more likely to connect with people they already know, or have some offline basis for the connection [6]. In this case, his/her friends tend to check-in to near PoIs to the query point. This suggests that distance between attribute and social attribute values have correlation. To this end, we design a sub-scoring function f'_S, which employs f_S of Eq. (5) and is employed instead of f_S in Eq. (3), as follows.

$$f'_S(q_u, n_i) = \begin{cases} f_S(q_u, n_i) \cdot {f_G}^d(q_u, n_i) & (n_i.G(q_u) \neq 0) \\ f_S(q_u, n_i) & (n_i.G(q_u) = 0) \end{cases}, \tag{7}$$

where d is a real number which provides the influence of distance attribute. Note that d is selected by some empirical studies.

Based on the above discussion, we finally redesign a scoring function F which we propose in this paper and is described below.

$$F(q_u, n_i) = \begin{cases} {f_G}^{d+1}(q_u, n_i) \cdot f_S(q_u, n_i) \cdot f_K(q_u, n_i) & (f_G(q_u, n_i) \neq 0) \\ -\frac{1}{f_S(q_u, n_i) \cdot f_K(q_u, n_i)} & (f_G(q_u, n_i) = 0) \end{cases}$$

This scoring function is based on Eqs. (3) and (7).

4 Experimental Evaluation

4.1 Experimental Setup

We used two real datasets of Gowalla and Brightkite detailed in Table 1. These datasets include: (1) PoIs which hold IDs and locations, (2) users who hold logs of friendships and check-ins. For each PoI in a given dataset, we assigned five synthetic keywords, and the number of distinct keywords in the dataset

Table 1. Datasets

Dataset	#Users	#PoIs	#Friendships	#Check-ins
Gowalla	196,591	1,280,969	1,900,655	3,981,334
Brightkite	58,228	772,965	428,157	1,072,965

is 10,000. To simulate semantic proximities (e.g., the keyword of "restaurant" tends to appear with the keyword of "lunch"), we used continuous five integers calculated by modulo 10,000 as the keywords (e.g., 9,999, 0, 1, 2, 3). For these datasets, we run 100 GSKSQs by the following way. We first pick a random user and choose 1–5 query keywords in the same way employed for PoIs. We calculate the MBR of PoIs which the user has checked-in, and choose a random location from the MBR as a query point. For α in Eqs. (5) and (6), we use 0.001. All algorithms were implemented in C++, and executed on Intel Xeon 3.47 GHz with 192 GB RAM.

4.2 Experimental Result

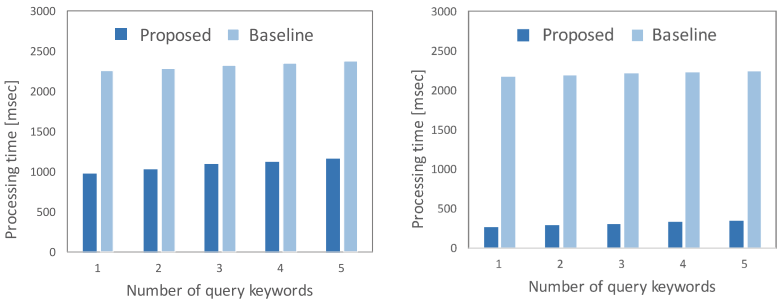
Pre-processing. The calculation times of upper-bound social values are only 67.7 s for Gowalla, and 35.2 s for Brightkite. These are trivial times in practice. The calculation for Gowalla takes more time because Gowalla has more users, check-ins, and friendships. In addition, the construction times of SKR-tree are 4.38×10^4 s for Gowalla, and 2.38×10^4 s for Brightkite. These are reasonable times in practice because we can update them once we construct them. Gowalla takes more time because it has more PoIs.

Results with Different Number of Query Keywords. We run GSKSQs with different number of query keywords to demonstrate the efficiency of our algorithm, and to investigate the impact of the number of query keywords. The result is shown in Fig. 4. We used the best d shown in Table 2, which are acquired from empirical experimental results. We can see that our algorithm significantly outperforms the baseline. It is also seen that both algorithms need longer processing times as the number of query keywords increases. When the number of query keywords is large, the number of PoIs which have the common keywords with query keywords is also large. Hence, the number of dominance checks increases.

Table 2. The best d for each number of query keywords

Dataset	Number of query keywords				
	1	2	3	4	5
Gowalla	101	116	122	122	122
Brightkite	119	119	119	120	120

We here show the pruning rate of our algorithm in Fig. 5. We define pruning rate as a rate of pruned nodes to the entire nodes. According to Fig. 5, the pruning rates for Gowalla are lower than those of Brightkite. This difference is explained by Table 1. The differences of the numbers of users, check-ins, and friendships are larger than the difference of the number of PoIs, and that causes differences of attribute values of social relationships. These pruning rates bring the results that the processing times of the proposed algorithm on Gowalla are in 43%–49% compared to that of the baseline one, while those on Brightkite are in 12%–16%.



(a) Gowalla (b) Brightkite

Fig. 4. Impact of the number of query keywords

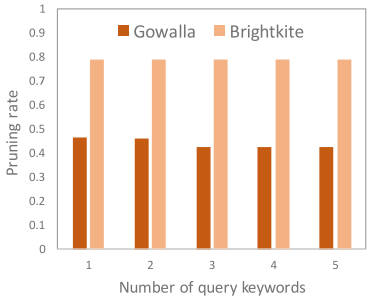


Fig. 5. Pruning rates for the two datasets

5 Conclusion

In this paper, we introduced a novel query, GSKSQ. Although this query supports decision making in LBSNS, it requires high processing cost when we use a

naive algorithm. To overcome this problem, we proposed an index called SKR-tree and an algorithm which uses this index. Our algorithm optimizes a number of node accesses, and achieves effective dominance checks by considering a practical characteristic of social relationships. The experimental results demonstrate that the performance of our algorithm is better than that of the baseline algorithm.

References

1. Ahuja, R., Armenatzoglou, N., Papadias, D., Fakas, G.J.: Geo-social keyword search. In: Claramunt, C., Schneider, M., Wong, R.C.-W., Xiong, L., Loh, W.-K., Shahabi, C., Li, K.-J. (eds.) SSTD 2015. LNCS, vol. 9239, pp. 431–450. Springer, Cham (2015). doi:10.1007/978-3-319-22363-6_23
2. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
3. Emrich, T., Franzke, M., Mamoulis, N., Renz, M., Züfle, A.: Geo-social skyline queries. In: Bhowmick, S.S., Dyreson, C.E., Jensen, C.S., Lee, M.L., Muliantara, A., Thalheim, B. (eds.) DASFAA 2014. LNCS, vol. 8422, pp. 77–91. Springer, Cham (2014). doi:10.1007/978-3-319-05813-9_6
4. Mouratidis, K., Li, J., Tang, Y., Mamoulis, N.: Joint search by social and spatial proximity. IEEE TKDE **27**(3), 781–793 (2015)
5. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP operations in spatial data warehouses. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 443–459. Springer, Heidelberg (2001). doi:10.1007/3-540-47724-1_23
6. Steinfield, C., Ellison, N., Lampe, C., Vitak, J.: Online social network sites and the concept of social capital. Front. New Media Res. **15**, 115–131 (2012)
7. Tan, K.-L., Eng, P.-K., Ooi, B.C., et al.: Efficient progressive skyline computation. In: VLDB, vol. 1, pp. 301–310 (2001)
8. Tao, Y., Sheng, C.: Fast nearest neighbor search with keywords. IEEE TKDE **26**(4), 878–888 (2014)