On Efficient Processing of Group and Subsequent Queries for Social Activity Planning

Yi-Ling Chen, De-Nian Yang, *Senior Member, IEEE*, Chih-Ya Shen, Wang-Chien Lee, *Member, IEEE*, and Ming-Syan Chen, *Fellow, IEEE*

Abstract—Three essential criteria are important for social activity planning: (1) finding attendees familiar with the initiator, (2) ensuring most attendees have tight social relations with each other, and (3) selecting an activity period available to all. In this paper, we propose the *Social-Temporal Group Query (STGQ)* to find suitable time and attendees with minimum total social distance. We first prove that the problem is NP-hard and inapproximable within any ratio. Next, we design two algorithms, *SGSelect* and *STGSelect*, which include effective pruning techniques to substantially reduce running time. Moreover, as users may iteratively adjust query parameters to fine tune the results, we study the problem of *Subsequent Social Group Query (SSGQ)*. We propose the *Accumulative Search Tree* and *Social Boundary*, to cache and index intermediate results of previous queries in order to accelerate subsequent query processing. Experimental results indicate that SGSelect and STGSelect are significantly more efficient than baseline approaches. With the caching mechanisms, processing time of subsequent queries can be further reduced by 50-75 percent. We conduct a user study to compare the proposed approach with manual activity coordination. The results show that our approach obtains higher quality solutions with lower coordination effort, thereby increasing the users' willingness to organize activities.

Index Terms—Social networks, query processing, indexing structure

1 Introduction

Two essential criteria for social activity planning are (1) finding a group of attendees familiar with the initiator and (2) ensuring most attendees have tight social relations with each other. For the social activities of which the time is not pre-determined, an additional essential criterion would be (3) selecting an activity period available to all attendees. For example, a person with some free movie tickets to share may like to find a group of mutually close friends and a time available to all. Nowadays, most activities are still initiated manually via phone, e-mail, or texting. However, with a growing number of systems possessing information needed for activity initiation, new activity planning functions can be provided. For example, social networking websites, such as Facebook, Google+, and LinkedIn, provide the social relations, and web applications, such as Google Calendar, Doodle,[1] NeedToMeet,[2] and Meetup,[3] allow people to share their available time and activity plans to friends. For manual activity planning, finding socially close participants and a suitable time can be tedious and time-consuming, due to the complexity of social connectivity and the diversity of schedules. Thus, there are demands for an effective activity planning service that automatically suggests socially acquainted attendees and a suitable time for an activity.

To support the aforementioned service, we first formulate a new query problem, named *Social Group Query (SGQ)*, which considers the relationships of the activity initiator and candidate attendees. Given an activity initiator, we consider her social network for candidate attendees, and the closeness between friends can be quantitatively captured as *social distance* [1], [2], [3]. Based on the criteria mentioned earlier, an SGQ comes with the following parameters, (1) a group size p to specify the number of attendees, (2) a social radius constraint s for the scope of candidate attendees, and (3) an acquaintance constraint k to govern the relationships between attendees. SGQ aims to find a group matching the group size in (1), such that the total social distance between the initiator and attendees is minimized. Additionally, the social radius constraint in (2) specifies that all attendees are located no more than s edges away from the initiator, while the acquaintance constraint in (3) requires that each attendee has at most k unacquainted attendees. As such, by controlling s and k based on the desired social atmosphere, suitable attendees are returned.

To support the planning service for the activities with no pre-determined time, we further propose another new

- *Y.-L. Chen is with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 10607, Taiwan. E-mail: yiling@mail.ntust.edu.tw.*
- *D.-N. Yang is with the Institute of Information Science, Academia Sinica, Taipei 11529, Taiwan. E-mail: dnyang@iis.sinica.edu.tw.*
- *C.-Y. Shen is with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan. E-mail: chihya@cs.nthu.edu.tw.*
- *W.-C. Lee is with the Department of Computer Science and Engineering, Pennsylvania State University, State College, PA 16801. E-mail: wlee@cse.psu.edu.*
- *M.-S. Chen is with the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan. E-mail: mschen@cc.ee.ntu.edu.tw.*

Manuscript received 29 Nov. 2016; revised 1 Feb. 2018; accepted 28 Mar. 2018. Date of publication 16 Oct. 2018; date of current version 4 Nov. 2019. (Corresponding author: Ming-Syan Chen)
Recommended for acceptance by W. Zhang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2018.2875911

1. The Doodle website. http://doodle.com/.
2. The NeedToMeet website. http://www.needtomeet.com/.
3. The Meetup website. http://www.meetup.com/.

query problem, named *Social-Temporal Group Query (STGQ)*, which considers both of the available time and relationships of candidate attendees. We assume that the schedules of candidate attendees are available to the planning service (e.g., via web collaboration tools). Besides the three parameters in SGQ, STGQ has a fourth parameter for the temporal dimension, which is (4) an availability constraint m to specify the length of activity period. STGQ aims to find a group and time matching the group size in (1) and the activity length in (4), such that the total social distance between the initiator and attendees is minimized. Moreover, the returned group of STGQ also satisfies the social radius constraint in (2) and the acquaintance constraint in (3). Example A.1 in Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2018.2875911, presents an illustrative example of SGTQ. Note that, while possessing the required information, all of the aforementioned web applications (e.g., Doodle) still cannot automatically find the suitable attendees and time for the initiators. To the best of our knowledge, the STGQ problem has not been studied before.

In this paper, we first examine the processing strategies for SGQ and then extend our study to the more complex STGQ. Solving an SGQ may incur an exponential time because SGQ is NP-hard, and processing an STGQ is even more challenging due to the diversity of schedules. When sequentially choosing the attendees at each iteration to form a candidate group, giving priority to close friends of the initiator may lead to a smaller total social distance. However, it may not necessarily satisfy the acquaintance constraint. On the other hand, prioritizing a set of mutually close friends to address the acquaintance constraint does not guarantee minimum total social distance. Moreover, in processing an STGQ, a group of mutually acquainted friends with a small total social distance still cannot form a solution if their available times do not overlap. Therefore, the challenge comes from the strategical dilemma between reducing the total social distance and ensuring that the solution follows the constraints both socially and temporally. Based on the above observations, we propose an algorithm called *SGSelect* that addresses both the social distance and connectivity, and then extend it to *STGSelect* by incorporating various strategies for the temporal dimension. Compared with the existing studies that focus on only the social dimension to find densely-connected subgroups (e.g., [4], [5], [6], [7], [8]), STGSelect can process both temporal and social dimensions effectively and efficiently.

Note that it is difficult for a user to specify all the query parameters right to find the perfect group of attendees and time. Fortunately, with SGQ and STGQ, it is easy for the user to tune the parameters to find alternative solutions. For example, the initiator may decrease k to tighten the group, or increase s to incorporate more friends of friends. Allowing tuning parameters to try subsequent queries is a great advantage of the planning service over the current practice of manual planning. However, it has not been explored in related works [4], [5], [6], [7], [8], [9], [10], [11], [12]. Some existing studies [13], [14], [15], [16] on subgraph queries return multiple subgraphs in one single diversified query. However, without feedback and guidance from user-specified parameters, most returned subgraphs are likely to be redundant (i.e., distant from the desired results of users). In realistic, users usually review the result obtained with the current parameter setting, and then adjust the parameter setting for subsequent queries.

A straightforward method to support a sequence of SGQs is to answer each individual query with SGSelect. However, anticipating that the users would not adjust the parameters drastically, we envisage that exploiting the intermediate solutions of previous queries may improve processing of subsequent queries. To facilitate the above idea, we propose *Subsequent Social Group Query (SSGQ)*, which aims to efficiently support a sequence of SGQs with varying parameters, and SSGQ can be extended to support a sequence of STGQs. Accordingly we design a new index structure, namely, *Accumulative Search Tree*, which caches the intermediate solutions of historical queries in a compact form for reuse. To facilitate efficient lookup, we further propose another new index structure, called *Social Boundary*, which effectively indexes the intermediate solutions required for processing each SSGQ with specified parameters. We devise various *node selection rules* to carefully select a sufficient and necessary set of candidate subgroups for extracting the final solution. We prove that the returned solution is optimal, even if SSGQ only processes a small portion of candidate subgroups in a social boundary.

Contributions of this paper are summarized as follows.

- We formulate two useful queries for social activity planning, namely, SGQ and STGQ, to obtain the optimal set of attendees and a suitable activity time. These queries can be used to plan for various activities by specifying the social radius constraint s and the acquaintance constraint k. We also prove these two problems are NP-hard and inapproximable within any ratio. That is, there exists no approximation algorithm for SGQ and STGQ unless P = NP.
- We propose Algorithms SGSelect and STGSelect to efficiently find the optimal solution to SGQ and STGQ, respectively. Moreover, we devise various strategies, including *access ordering*, *distance pruning*, *acquaintance pruning*, *pivot time slots*, and *availability pruning*, to prune redundant search space and improve efficiency. Our research results can support social networking websites and web collaboration tools as a value-added service.
- To support tuning of query parameters to find more desirable results, we propose SSGQ and design two new index structures, *accumulative search tree* and *social boundary*, to accelerate a sequence of group queries. The proposed index structures effectively cache and index intermediate solutions in preceding queries to support subsequent ones. Experimental results show that the caching mechanisms substantially reduce processing time.
- We conduct a user study to compare the proposed planning service with manual coordination, and collect feedbacks as a guidance to enrich our group query service. The results show that the proposed algorithms are able to obtain higher solution quality with much lower coordination effort, increasing users' willingness to organize activities.

The rest of this paper is summarized as follows. In Section 2, we introduce related works. In Section 3, we then formulate SGQ, explain the details of Algorithm SGSelect, and extend SGSelect to STGSelect for STGQ. After that, we

introduce SSGQ and the caching mechanisms used to support the subsequent queries in Section 4. We then detail the construction and maintenance of the caching mechanisms and proof the solution optimality in Section 5. The SSGQ is extended to support the temporal dimension in Section 6. Finally, we present the experimental results in Section 7 and conclude this paper in Section 8.

2 Related Work

Although some web applications have been developed to support activity planning and coordination, they require users to manually identify the activity time and participants. For example, with the Events function on Facebook, an activity initiator can specify an activity time and select friends to invite. These friends then reply with whether they can attend or not. Some popular activity/event planning websites and apps, such as Doodle,[4] Pick,[5] Rallly,[6] UFr.ee,[7] Boomerang Calendar,[8] Time Zone Ninja,[9] MeetOMatic,[10] WhenIsGood,[11] and NeedToMeet,[12] are developed to help an activity initiator to collect the available times of potential event participants in order to find a suitable activity time slot and choose an appropriate group for the activity. More details of these services are provided in Appendix A.2, available in the online supplemental material.

Note that the aforementioned services are quite popular and have been widely adopted in everyday life for activity organization. For example, Doodle serves 20 million users per month,[13] and NeedToMeet has 30 million users.[14] However, these apps still have space for improvement. Notice that the initiator needs to manually decide a desired activity time and the corresponding participants, especially when the target social atmosphere is considered. As a research to explore new ideas, in this paper we propose STGQ to complement the existing exercises in the aforementioned web applications in order to automatically find a group of close friends for planning some social event at a suitable activity time. The result of a user study (see Appendix E.4 , available in the online supplemental material) confirms that STGQ obtains higher quality solutions with much lower coordination efforts, thereby increasing users' willingness to foster different kinds of activities.

By minimizing the total social distance among the attendees, we aim at forming a cohesive subgroup in the social network. Related research on extracting dense subgraphs includes the clique, k-plex, k-core, k-truss, k-clan, and k-club [4], [5], [6], [7], [8], [17], [18], [19], [20], [21], [22], [23], [24]. However, most of them focus on enumerating maximal cohesive subgroups or finding the maximum cohesive subgroup. For example, the k-core [17], [18] finds the largest subgraph with the degree of every vertex at least k within the subgraph. However, the group sizes of the returned solutions in the above studies are not controllable, and this size tends to be relatively large when the social network is not sparse. In contrast, the proposed SGQ allows the activity initiator to specify the desired group size, whereas the temporal dimension is further examined in STGQ.

Moreover, group formation [1], [25], [26], [27], team formation [9], [10], [11], group query [12], [28], [29], community search and social circle discovery [30], [31], [32] have been studied for various applications. More specifically, long-term evolution of social groups in global social networks has been analyzed in [1]. Group formation [25], [26], [27] aims to find a group of members with high similarity on a certain activity topic. Team formation [9], [10], [11] focuses on finding a group of experts covering all required skills and minimizing the total communication cost. Moreover, group queries [12], [28], [29], are designed to find a socially cohesive group. Community search [30] extracts a compact community with the objective to maximize the minimum degree of certain nodes in the community. In reality, however, there are various types of activities with different social atmosphere requirements. Therefore, instead of maximizing the minimum degree, we allow activity initiators to specify the suitable social constraints s and k based on their desired social atmosphere accordingly. Another types of community search [31] aims to identify effective structures to detect the densely linked communities, and the social circle discovering model [32] extracts social circles of members having common properties. However, the sizes of communities and social circles are not restricted. Moreover, none of them incorporates the temporal dimension to find a suitable activity time, and subsequent queries with different parameter settings are not considered in the above work.

On the other hand, diversified queries [13], [14], [15], [16] return multiple subgraphs with diverse graph characteristics. However, since these studies are not specifically designed for activity planning, the social tightness is not ensured, and many returned subgraphs in the diversified query are inclined to miss the user requirements on social atmosphere. On the other hand, session query and reinforcement learning in retrieval (e.g., [33], [34], [35]) that allow users to tailor the query have attracted increasing attentions. However, these studies are designed for document retrieval, instead of social graphs and user schedules. Therefore, these aforementioned research works are not applicable for automatic activity planning.

3 Social Group Query

In Section 3.1, we present the problem formulation and research challenges of SGQ, and prove that SGQ is NP-hard. We then propose an algorithm that effectively prunes redundant search space to obtain the optimal solution in Section 3.2. Finally, we extend SGQ to STGQ in Section 3.3.

3.1 Problem Definition

Given an activity initiator q and her social graph $G = (V, E)$, where each vertex v is a candidate attendee, and the distance on each edge $e_{u,v}$ connecting vertices u and v represents their social closeness. A social group query $SGQ(p, s, k)$, where p is a group size, s is a social radius

4. The Doodle website. http://doodle.com/.
5. The Pick website. http://www.pick.co/.
6. The Rallly website. http://rallly.co/.
7. The UFr.ee website. http://ufr.ee/.
8. The Boomerang Calendar website. http://boomerangcalendar.com/.
9. The Time Zone Ninja website. http://timezoneninja.com/.
10. The MeetOMatic website. http://www.meetomatic.com/.
11. The WhenIsGood website. http://www.whenisgood.net/.
12. The NeedToMeet website. http://www.needtomeet.com/.
13. The statistics are reported on the Doodle website. http://doodle.com/press/milestones/.
14. The statistics are reported on the NeedToMeet website. http://www.needtomeet.com/pdf/NeedToMeet%20Add-in%202.0%20Release%20Notes.pdf.

constraint, and k is an acquaintance constraint, finds a set F of p vertices from G to minimize the total social distance between q and every vertex v in F, i.e., $\sum_{v \in F} d_{v,q}$, where $d_{v,q}$ is the length of the minimum-distance path between v and q with at most s edges, such that each vertex in F is allowed to share no edge with at most k other vertices in F.

The initiator can specify different s for various activities. Specifying $s = 1$ ensures direct acquaintance between the initiator and attendees, while $s > 1$ allows for friends of friends. The initiator can also vary k for different activities, using a small k for intimate gathering or a large k for a more diverse event. Indeed, processing SGQ is NP-hard and inapproximable within any ratio. In other words, there is no approximation algorithm with a finite ratio unless P = NP. Fortunately, while challenging, the problem is still tractable when the size of s and p are reasonable.

Theorem 1. *SGQ is NP-hard and inapproximable within any ratio.*

Proof. We prove the theorem in Appendix D.1, available in the online supplemental material. □

3.2 Algorithm Design

In this section, we propose an algorithm, SGSelect, to efficiently solve SGQ. Our idea is to first derive a *feasible graph*$G_F = (V_F, E_F)$ from G, such that there exists a path with at most s edges from q to each vertex in G_F. Starting from q, we then iteratively explore G_F to derive the optimal solution. At each iteration, we keep track of the set of vertices that satisfy the acquaintance constraint as the intermediate solution obtained so far (denoted as V_S). Initially, we set $V_S = \{q\}$, and let V_A denote the set of remaining vertices in V_F, i.e., $V_A = V_F - V_S$. We select a vertex in V_A and examine whether it is feasible (i.e., follows the acquaintance constraint) to move this vertex to V_S at each iteration, continuing until $|V_S| = p$.

In constructing candidate groups, judicious access order of vertices is crucial. It is essential to avoid vertices that significantly increase the total social distance or lead to a violation of the constraint. It is also important to prioritize vertices likely to form high-quality feasible solutions, to facilitate effective early pruning. Additionally, social radius and acquaintance constraints can be exploited to prune unqualified vertices. Finding the optimal solution to an SGQ may incur an exponential time because SGQ is NP-hard. However, by employing the proposed strategies, the average running time of Algorithm SGSelect can be effectively reduced, as to be shown in Section 7. In the following, we present the details of the proposed algorithm.

3.2.1 Radius Graph Extraction

Algorithm SGSelect first extracts the vertices that satisfy the social radius constraint to prune redundant candidates. A simple approach is to find the *minimum-edge path* (i.e., the shortest path with the minimum number of edges) between q and every other vertex, and then remove the vertices that have minimum-edge paths with more than s edges. To address this, we define the notion of *i-edge minimum distance*, which represents the total distance of the minimum-distance path with no more than i edges. The i-edge minimum distance between two vertices v and q is $d^i_{v,q} = \min_{u \in N_v}\{d^{i-1}_{v,q}, d^{i-1}_{u,q} + c_{u,v}\}$, where N_v is the set of neighboring vertices of v, and $c_{u,v}$ is the weight of edge $e_{u,v}$. Based on dynamic programming, the i-edge minimum distance between v and q is computed by iteratively deriving $d^i_{v,q}$ in terms of $d^{i-1}_{u,q}$ of each neighboring vertex u, for $1 \le i \le s$. After that, each vertex v with $d^s_{v,q} < \infty$ is extracted to construct a feasible graph $G_F = (V_F, E_F)$, and $d^s_{v,q}$ is adopted as the social distance between v and q. In the feasible graph, every vertex is guaranteed to satisfy the social radius constraint, and we consider G_F in evaluating the SGQ for the rest of this paper.

3.2.2 Access Ordering

After constructing G_F, Algorithm SGSelect iteratively explores it to find the optimal solution. Initially, we set $V_S = \{q\}$ and $V_A = V_F - \{q\}$. At each iteration afterwards, we select and move a vertex from V_A to V_S to expand the intermediate solution. The set V_S represents a feasible solution when $|V_S| = p$ and its vertices satisfy the acquaintance constraint. Next, our algorithm improves the feasible solution by backtracking the above exploration procedure to previous iterations and choosing an alternative vertex in V_A to expand V_S. A branch-and-bound tree is maintained to record the exploration history.

To reduce running time, the selection of a vertex at each iteration is critical. Naturally, we would like to include a vertex with the smallest social distance. However, the connectivity of the selected vertex imposes additional requirements for satisfying the acquaintance constraint. Thus, we introduce interior unfamiliarity condition and exterior expansibility condition to test the feasibility of examined vertices to the acquaintance constraint. When expanding V_S, SGSelect prioritizes the vertex that has the minimum social distance and satisfies both the interior unfamiliarity and exterior expansibility conditions.

Definition 1. *The interior unfamiliarity of V_S is*

$$U(V_S) = \max_{v \in V_S} |V_S - \{v\} - N_v|,$$

where N_v is the set of neighboring vertices of v in G_F.

The interior unfamiliarity describes the connectivity within V_S, with a smaller value representing more densely connected vertices. In generating candidate groups, it is preferable to first include a well-connected vertex that produces an intermediate solution set with low $U(V_S)$ to facilitate selections of other vertices in later iterations. Based on this observation, we leverage the *interior unfamiliarity condition*, i.e., $U(V_S \cup \{v\}) \le k\lceil\frac{|V_S \cup \{v\}|}{p}\rceil^\theta$, where $\theta \ge 0$ and $\frac{|V_S \cup \{v\}|}{p}$ is the proportion of attendees that have been considered, to ensure that the interior unfamiliarity value remains small when the vertex v is selected. Note that the right-hand-side (RHS) of the inequality reaches its maximum (i.e., k) when θ is fixed as 0. With $\theta = 0$, there is flexibility in finding a vertex v with a small social distance. However, if a vertex v resulting in $U(V_S \cup \{v\}) = k$ is selected, the vertex with k non-neighboring vertices in the set $V_S \cup \{v\}$ is required to connect to all future vertices chosen. This decreases the feasibility of selecting other qualified vertices in later iterations. In contrast, a larger θ allows SGSelect to choose a vertex from V_A that connects to more vertices in V_S to ensure the feasibility at later iterations. The algorithm reduces θ if there exists no vertex in V_A that can satisfy the above condition. When θ decreases to 0 and the above condition still does not hold (i.e., $U(V_S \cup \{v\}) > k$), Algorithm SGSelect stops

expanding the new intermediate solution set $V_S = V_S \cup \{v\}$, because adding any vertex from V_A does not generate a feasible solution, as shown by the following lemma.

Lemma 1. *Given that*

$$U(V_S) > k, \tag{1}$$

there must exist at least one vertex v in V_S violating the acquaintance constraint for every possible selection of vertices from V_A.

Proof. If $U(V_S) > k$, then we can find at least one vertex v in V_S such that $|V_S - \{v\} - N_v| > k$, which means v is already unacquainted with more than k other vertices in V_S. Note that adding a vertex from V_A to V_S does not increase the connectivity between the existing vertices in V_S. When we expand V_S by adding any vertex u from V_A, the number of vertices that are unacquainted with v remains unchanged if u and v are connected, and increases by one if not. That is, v still has more than k unacquainted vertices in V_S, which violates the acquaintance constraint. The lemma follows. □

Definition 2. *The exterior expansibility of V_S is*

$$A(V_S) = \min_{v \in V_S} \{|V_A \cap N_v| + (k - |V_S - \{v\} - N_v|)\}, \tag{2}$$

where the first set (i.e., $V_A \cap N_v$) contains the neighboring vertices of v in V_A and the second set (i.e., $V_S - \{v\} - N_v$) contains the non-neighboring vertices of v in V_S.

The exterior expansibility counts the number of options when selecting vertices from V_A to expand V_S, and a larger value represents more options satisfying the acquaintance constraint. Specifically, since the number of existing non-neighboring vertices of v in V_S is $|V_S - \{v\} - N_v|$, we can select at most $k - |V_S - \{v\} - N_v|$ extra non-neighboring vertices of v from V_A to expand V_S; otherwise, vertex v would have more than k non-neighboring vertices in V_S and violate the acquaintance constraint. Therefore, there are at most $|V_A \cap N_v|$ neighboring vertices and $k - |V_S - \{v\} - N_v|$ non-neighboring vertices can be selected, and their summation must be no smaller than the number of attendees required to be added later (i.e., $p - |V_S|$). Therefore, SGSelect chooses the vertex v from V_A that satisfies the *exterior expansibility condition*, i.e., $A(V_S \cup \{v\}) \geq (p - |V_S \cup \{v\}|)$. If the inequality does not hold, the new intermediate solution set obtained by adding v is not expansible, as shown by the following lemma.

Lemma 2. *Given that*

$$A(V_S) < (p - |V_S|), \tag{3}$$

there must exist at least one vertex v in V_S such that v cannot follow the acquaintance constraint for every possible selection of vertices from V_A.

Proof. If $A(V_S) < (p - |V_S|)$, there is at least one vertex v in V_S such that $|V_A \cap N_v| + (k - |V_S - \{v\} - N_v|) < (p - |V_S|)$. Since $|V_S - \{v\} - N_v|$ is the number of non-neighboring vertices for v, $k - |V_S - \{v\} - N_v|$ represents the "quota" for v to choose non-neighboring vertices from V_A. For any possible selection $\widehat{V}_A \subseteq V_A$, let $\widehat{\lambda}_A$ denote the number of neighboring vertices of v in $\widehat{V}_A$. Since $\widehat{\lambda}_A \leq |V_A \cap N_v|$, $(p - |V_S|) - |V_A \cap N_v| \leq (p - |V_S|) - \widehat{\lambda}_A$. We can derive that $(k - |V_S - \{v\} - N_v|) < (p - |V_S|) - \widehat{\lambda}_A$, meaning any possible selection exceeds the quota for v and violates the acquaintance constraint. The lemma follows. □

3.2.3 Distance and Acquaintance Pruning

In the following, we exploit two pruning strategies to reduce search space. Our algorithm aims to obtain a feasible solution early since the total social distance of this solution can be used for pruning redundant candidates. At each iteration, the distance pruning strategy avoids the vertices in V_A that do not lead to a solution with a smaller total social distance.

Lemma 3. *The distance pruning strategy stops selecting a vertex from V_A to V_S if*

$$D - \sum_{v \in V_S} d_{v,q} < (p - |V_S|) \min_{v \in V_A} d_{v,q}, \tag{4}$$

where D is the total social distance of the best feasible solution obtained so far. The distance pruning strategy can prune the search space with no better solution.

Proof. If the above condition holds, the total social distance of any new solution must exceed D when we select $p - |V_S|$ vertices from V_A, which means exploring V_A yields no better solution. As D improves at later iterations, we are able to derive a tighter bound on the LHS, and thus prune a larger search space. The lemma follows. □

We also propose an acquaintance pruning strategy that considers the connectivity of vertices in V_A. Note that all vertices in V_A can be excluded from expansion if $\sum_{v \in V_A} |V_A \cap N_v| < (p - |V_S|)(p - |V_S| - k - 1)$. The LHS of the inequality is the total inner degree of all vertices in V_A, where the *inner degree* of a vertex in V_A considers only the edges connecting to other vertices in V_A. The RHS corresponds to the minimum required value of the total inner degree on any set of vertices extracted from V_A that can expand V_S into a solution satisfying the acquaintance constraint. This strategy can be further improved by replacing the LHS with $\sum_{v \in M_A} |V_A \cap N_v|$, where M_A denotes the set of $p - |V_S|$ vertices in V_A with the largest inner degrees. Since $\sum_{v \in M_A} |V_A \cap N_v| \leq \sum_{v \in V_A} |V_A \cap N_v|$, our algorithm is able to prune off more infeasible solutions. Specifically, the acquaintance pruning is specified as follows.

Lemma 4. *The acquaintance pruning strategy stops selecting a vertex from V_A to V_S if*

$$\sum_{v \in M_A} |V_A \cap N_v| < (p - |V_S|)(p - |V_S| - k - 1), \tag{5}$$

and the acquaintance pruning strategy can prune the search space with no feasible solution.

Proof. Since we will only extract $p - |V_S|$ vertices from V_A to join V_S, the upper bound of total inner degree of the extracted vertices is $\sum_{v \in M_A} |V_A \cap N_v|$. If these $p - |V_S|$ extracted vertices follow the acquaintance constraint, each of them must be acquainted with at least $p - |V_S| - k - 1$ extracted vertices, which means the inner degree will be at least $(p - |V_S|)(p - |V_S| - k - 1)$. When the acquaintance pruning happens, even the upper bound of total inner degree of the extracted vertices is smaller than $(p - |V_S|)$

$(p - |V_S| - k - 1)$, which indicates that there is at least one vertex unacquainted with more than k vertices. Therefore, the pruned search space contains no feasible solution. The lemma follows. □

In the following, Theorem 2 proves that Algorithm SGSelect with the above strategies finds the optimal solution. Moreover, Example A.2 in Appendix A.1, available in the online supplemental material, provides an illustration of Algorithm SGSelect.

Theorem 2. *SGSelect obtains the optimal solution to SGQ.*

Proof. We prove the theorem in Appendix D.3, available in the online supplemental material. □

3.3 Extensions in Temporal Dimension

Social-Temporal Group Query (STGQ) generalizes SGQ by considering each candidate attendee's available time via the *availability constraint*, which ensures that all selected attendees are available for the activity period. Specifically, given the social graph G of an initiator q, a social-temporal group query $STGQ(p, s, k, m)$ finds a time slot t and a set F of p vertices from G to minimize the total social distance between q and every vertex in F, i.e., $\sum_{u \in F} d_{u,q}$, where $d_{u,q}$ is the length of the minimum-distance path between u and q with at most s edges, such that each vertex u in F is allowed to share no edge with at most k other vertices in F, and u is available from time slot t to $t + m - 1$.

STGQ is an NP-hard problem because STGQ can be reduced to SGQ if every candidate attendee is available in all time slots. An intuitive approach to evaluate STGQ is to sequentially explore each time slot t and the candidate attendees available from t to $t + m - 1$ (i.e., m consecutive time slots). However, running time grows significantly when the number of time slots increases. Therefore, we devise Algorithm STGSelect, which explores the following features in the temporal dimension to reduce running time.

Pivot Time Slot. A time slot is a *pivot time slot* if its ID is im, where i is a positive integer. Note that any feasible solution to STGQ must include one pivot time slot; otherwise, the returned activity period would be shorter than m and violate the availability constraint. Therefore, instead of considering every interval from t to $t + m - 1$ for each time slot t, our algorithm leverages the pivot time slots to efficiently explore the solution space. For each pivot time slot im, Algorithm STGSelect extends SGSelect by considering the temporal information when selecting a vertex from V_A. Specifically, let T_S denote the set of consecutive time slots available to all vertices in V_S, and T_S must contain slot im. After certain iterations, if V_S includes p vertices satisfying the acquaintance constraint, and $|T_S| \geq m$, V_S and T_S together form a feasible solution.

Temporal Extensibility. When selecting a vertex from V_A, Algorithm STGSelect considers not only the social distance but also the temporal availability of the vertex to avoid vertices that lead to a small increase of the total social distance but end up disqualified by the availability constraint. Therefore, in addition to the interior unfamiliarity and exterior expansibility discussed in Section 3.2, we further consider the *temporal extensibility* of V_S, which is defined as $X(V_S) = |T_S| - m$. A larger temporal extensibility implies that more vertices in V_A of good quality can later be selected by our algorithm.

Availability Pruning. The *availability pruning* strategy enables our algorithm to stop exploring V_A if there exists no solution that can satisfy the availability constraint. Specifically, for each pivot time slot im, let $\overline{t}_A^+(n)$ and $\overline{t}_A^-(n)$ denote the time slots closest to im, such that at least n vertices in V_A are not available in the two time slots, where $\overline{t}_A^+(n) > im$ and $\overline{t}_A^-(n) < im$. The availability pruning strategy stops considering V_A when

$$\overline{t}_A^+(|V_A| - p + |V_S| + 1) - \overline{t}_A^-(|V_A| - p + |V_S| + 1) \leq m.$$

In this case, there are at most $p - |V_S| - 1$ vertices of V_A available in each of the above two slots, and the interval starting from $\overline{t}_A^- + 1$ to $\overline{t}_A^+ - 1$ contains fewer than m time slots. Consequently, we never find a feasible solution with this V_A because Algorithm STGSelect is required to choose $p - |V_S|$ vertices from V_A for a common available interval with at least m time slots.

To find the optimal solution, STGSelect is expected to have an exponential-time complexity because STGQ is NP-hard. In the worst case, all candidate groups in all time slots may need to be considered. However, as shown by the experimental results, the average running time of the proposed algorithm with the above strategies can be effectively reduced.

4 Subsequent Social Group Query

In real situations, users usually prefer to adjust the query parameters in order to retrieve better results. Moreover, according to the feedback from a user study (see Appendix E.4, available in the online supplemental material), initiators are inclined to adjust social constraints to consider varying recommendations, thus tend to issue a sequence of social group queries. Nevertheless, the above important need is largely ignored in related works [4], [5], [6], [7], [8], [9], [10], [11], [12]. A naive method to process the subsequent SGQs is to solve each SGQ independently with SGSelect. However, as these SGQs are fine-tuned by the same initiator with slight changes in parameters, we can improve the efficiency of subsequent SGQs by caching intermediate solutions for reuse. Therefore, we propose two mechanisms, *Accumulative Search Tree (AST)* and *Social Boundary (SB)*, to support processing of *Subsequent Social Group Queries (SSGQs)*. Instead of repeating the traversal for each individual SGQ, AST caches the intermediate solutions for different parameters in a single tree to process a sequence of SGQs from the same initiator. SBs then index the nodes in AST to facilitate lookup in the processing of new SGQs. Since the nodes not indexed by any SB are not used in subsequent SGQs, a large portion of nodes in AST can be discarded, significantly reducing caching overhead and processing cost. Moreover, duplicate searches are reduced because AST and SB enable the query processing of subsequent SGQs to start with intermediate solutions. In the following, Section 4.1 first introduces AST and SB. Section 4.2 then presents how to exploit AST and SB to answer SSGQs efficiently. We then detail the construction and maintenance of the caching mechanisms and proof the solution optimality in Section 5. After that, we will extend SSGQ to support the temporal dimension in Section 6.

4.1 Accumulative Search Tree and Social Boundary

Fig. 1 illustrates two search trees T_1 and T_2 corresponding to two SGQs with slightly different parameters $(s, k) = (2, 1)$

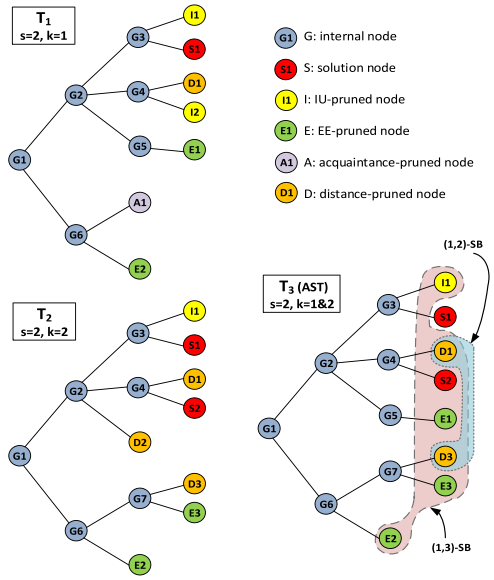


Fig. 1. T_1 and T_2 are two dendrograms with different k, and T_3 is the accumulative search tree.

and $(2,2)$, respectively. It can be observed that T_1 and T_2 share many nodes in common, including $G1-G4$, $G6$, $I1$, $S1$, $D1$, and $E2$. Nevertheless, some nodes are different due to various pruning strategies. For example, $G5$ in T_1 does not appear in T_2 due to the distance pruning (thus it is marked as $D2$ in T_2). Meanwhile, $G7$ in T_2 does not appear in T_1 because of the acquaintance pruning (thus it is marked as $A1$ in T_1). In addition to the distance pruning and acquaintance pruning, the interior unfamiliarity condition with $\theta = 0$ and the exterior expansibility condition also avoid traversing redundant branches, and here we refer them as *interior unfamiliarity pruning (IU pruning)* and *exterior expansibility pruning (EE pruning)*, respectively. It is important in the design of AST to cache not only the common parts but also the different parts of T_1 and T_2, in order to support different SGQs with a variety of query parameters subsequently in the future. Also, to support quick traversal of the tree in the subsequent queries, we propose SB to index the nodes in AST. For example, indexing $I1$ in T_1 allows future queries with parameter $k = 3$ to start with this node, instead of the root $G1$, to avoid traversing unnecessary nodes. In the following, we first introduce AST to cache the intermediate results of historical queries in a compact way.

Definition 3. *An accumulative search tree is a tree structure that includes (1) internal nodes (i.e., the nodes successfully expanded in historical queries), (2) pruned nodes (i.e., the nodes where prunings happen and act as the roots of pruned branches), and (3) solution nodes. Each tree node contains the information generated during query processing, such as V_S and V_A.*

The initial AST is the search tree generated in the first SGQ. Taking Fig. 1 as an example, the first query is with parameters $(s,k) = (2,1)$, and the initial AST is T_1, where $A1$ is a pruned node since there is a branch pruned by acquaintance pruning. Nodes Gi and Si stand for an internal node and a solution node, respectively. When processing the subsequent query, AST is updated by replacing the pruned node with an internal node to explore the branch not considered in the previous query. For example, the second query is with parameters $(s,k) = (2,2)$, and $A1$ in T_1 is replaced by $G7$ in T_3, implying that the previously pruned branch is explored in the new query.

When the user specifies a tighter constraint, such as $k = 0$, not all the nodes in the existing AST (i.e., T_3) are feasible for the tight constraint. On the other hand, although the root node is always feasible, it is not efficient to start the query processing with $G1$ because it leads to duplicate traversal. Therefore, it is desirable to index the nodes of T_3 for different social constraints in order to support subsequent queries, and we propose SB to address this issue.

Definition 4. *An (s_b, k_b)-social boundary contains pointers to a list of nodes in AST to accelerate the processing of the query with $s = s_b$ and $k = k_b$, such that expanding the nodes in the list leads to the optimal solution to this query.*

Note that, during the construction of the SBs for different s and k, nodes that cannot lead to the optimal solution are excluded.[15] While a pruned node may be included in an SB with a larger k for re-expansion, it may be excluded from another SB with a smaller k due to violating the tighter acquaintance constraint. In Fig. 1, there are two dashlined regions in T_3 representing (1,2)-SB and (1,3)-SB, respectively. There are only two nodes in (1,2)-SB, since the other five nodes in (1,3)-SB (i.e., $I1$, $S2$, $E1$, $E2$, and $E3$) violate the acquaintance constraint with $k = 2$ and thus are excluded from (1,2)-SB. Therefore, to answer a new query with $(s,k) = (1,2)$, we only need to expand the two nodes in (1,2)-SB, instead of all the 15 nodes in T_3. Specifically, the SBs can be viewed as a table containing pointers to a set of nodes, as shown in Table 1 with the query example in Fig. 2. The content of this table is filled using the nodes in AST after the first SGQ is processed. For each subsequent SGQ with specified s and k, we are able to simply extract the nodes in the corresponding (s,k)-SB and expand these nodes to find the optimal solution. These nodes in the SB can be treated as shortcuts on AST, where expanding them directly can avoid traversing from the root of AST to reduce the computation cost for new queries.

Finding the correct nodes for each SB is crucial due to the following reasons. First, if the SB contains some nodes too close to the root, it still needs to traverse some redundant internal nodes in AST, leading to duplicate exploration. Second, while a node close to the leaf nodes lowers traversal cost, the new branch expanded from this node only covers a small portion of the solution space, and the optimal solution is thereby not guaranteed. Third, if the SB includes the nodes that do not generate feasible solutions under the new social constraints, it incurs redundant caching overhead and computation cost. Therefore, it is important to select a sound and complete set of nodes for each SB. In the following, we first present how to efficiently acquire the solution by leveraging AST and SB in Section 4.2. We will then detail the construction of SBs in Section 5 and prove in Theorem 3 that the nodes indexed by (s,k)-SB are sufficient for finding the optimal solution.

4.2 Solution Acquisition Using AST and SB

For each new SSGQ with a specified pair of s and k, we first use the (s,k)-SB to quickly identify the corresponding nodes

15. The range of possible s and k are $1 \le s \le s_{max}$ and $0 \le k \le p-1$, where s_{max} is the largest possible s. According to the small world phenomenon [36], s_{max} does not need to be very large (e.g., 4), and users can also specify a desired s_{max}.

TABLE 1
The (s, k)-SBs Constructed after the First SGQ

(s, k)	Nodes indexed by the (s, k)-SB
...	...
(2,0)	P23, P27, P28, P29
(2,1)	P20, P23, P27, P28, P29 [S1]
(2,2)	P4, P11, P16, P18, P20, P22, P23, P27, P28, P29 [S1]
(2,3)	P2, P4, P9, P11, P16, P18, P20, P22, P23, P27, P28, P29 [S2]
(2,4)	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29 [S2]
(2,5)	...
(2,6)	...
(3,0)	P23, P27, P28, P29
(3,1)	P20, P23, P27, P28, P29 [S1]
(3,2)	P4, P11, P16, P18, P20, P22, P23, P27, P28, P29 [S1]
(3,3)	[already processed]
(3,4)	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29 [S2]
(3,5)	...
(3,6)	...
...	...

in AST. Each node indexed by the (s, k)-SB is then expanded to see if better solutions can be generated. The expansion here is similar to the recursive function $ExpandSG$ in Algorithm SGSelect (pseudo code provided in Appendix F, available in the online supplemental material): repeatedly expand the V_S of the node by selecting and adding the vertices from V_A to V_S, and then detect if the expanded V_S satisfies the acquaintance constraint. Newly generated solutions are compared with the existing one stored in the (s, k)-SB to choose that with the smallest total social distance. While processing the SSGQ, if any pruned node in the AST is successfully expanded, it will be replaced by the branches generated during the expansion. To update the SBs, the node removed from the AST is also eliminated from the SBs to avoid duplicate expansion. Nodes in the newly generated branches, on the other hand, are incorporated into SBs for reuse. More details about selecting the correct set of nodes for each SB will be provided later in Section 5.

Example 1. In this example, v_7 in Fig. 2a is the initiator. After processing the first query with $(p, s, k) = (7, 3, 3)$ using SGSelect, the initial AST is shown in Fig. 2b. Nodes Gi, Pi and Si stand for an internal node, a pruned node and a solution node, respectively. In Table 1, we list SBs for different s and k, with the details of its construction to be provided in Section 5. Assume that the initiator modifies the constraint k from 3 to 2 for a tighter group. Instead of examining all 43 nodes in the initial AST to obtain the optimal solution, with the help of (3,2)-SB, we only need to consider 11 nodes, which include 10 pruned nodes and one solution node $S1$. Each pruned node has its V_S and V_A, e.g., $V_S = \{v_2, v_6, v_7, v_8, v_9, v_{11}\}$ and $V_A = \{v_1, v_4, v_5, v_{10}\}$ for $P1$. We examine these pruned nodes to see if they can be successfully expanded to generate solutions, with a procedure similar to that in SGSelect. Note that since there is an existing solution $S1$ in (3,2)-SB, the distance pruning strategy is effective at early stages in the expansion and saves computation. If a pruned node is successfully expanded, it becomes an internal node that may lead to new solutions and will be replaced by the newly generated branches. The updated AST is shown in Fig. 2c. For example, a distance-pruned node $P4$ in Fig. 2b is successfully expanded into the internal node $G10$ in Fig. 2c, which eventually is expanded into a new solution node $S6$. After processing all the 10 pruned nodes in (3,2)-SB, we obtain four new feasible solutions (i.e., $S6$, $S7$, $S8$ and $S9$). Among them and the existing solution (i.e., $S1$), $S8$ is returned as the optimal one since it has the smallest total social distance.

5 Index Construction and Maintenance

Section 4 has illustrated how to exploit AST and SB to answer SSGQs. In the following, we further detail how to construct a table of (s, k)-SBs for various s and k. To effectively reduce redundant node processing in SSGQs, it is crucial to create SBs with the minimum number of nodes and ensure solution optimality by considering each kind of node (i.e., pruned nodes, solution nodes, and internal nodes) in

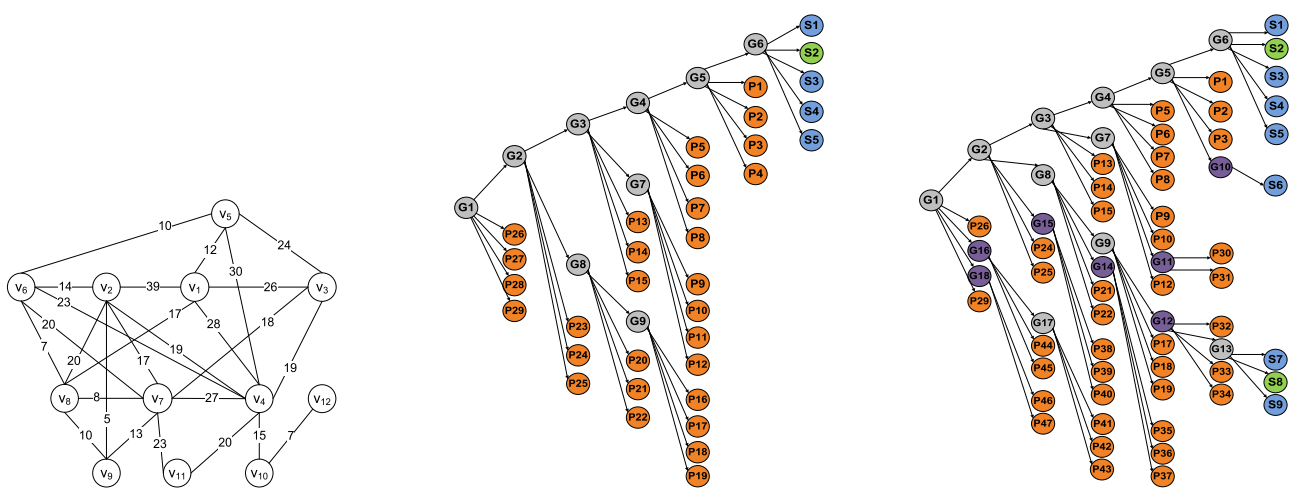


Fig. 2. An illustrative example for SSGQ. (a) The sample social network, (b) the initial accumulative search tree, and (c) the accumulative search tree after the second query.

AST. Here we address this essential issue by deriving a set of *node selection rules* for building SBs under various query parameters. We first focus on the acquaintance constraint k and then return to the social radius constraint s.

Rule 1: node indexing for different k

(1) Pruned nodes. We categorize pruned nodes into four types: *IU-pruned nodes*, *EE-pruned nodes*, *acquaintance-pruned nodes* and *distance-pruned nodes*, which correspond to Eqs. (1), (3), (5) and (4) in Section 3.2, respectively. Given s and k of the first SGQ, we examine if a pruned node is needed in the (s, k')-SB for processing a new SGQ with k' as follows.

- *IU-pruned nodes.* All IU-pruned nodes do not appear in any (s, k')-SB with $k' \leq k$, since k' represents a tighter acquaintance constraint. On the other hand, when $k' > k$, an IU-pruned node is not included in any (s, k')-SB if $k' < U(V_S)$ since insufficient social tightness within V_S prevents this node from becoming a solution. Therefore, an IU-pruned node only appears in the (s, k')-SB where

$$k' \geq max\{U(V_S), k+1\}. \tag{6}$$

  Example 2. Fig. 2b presents an illustrative example with an IU-pruned node $P1$ to identify the corresponding SBs. $P1$ is generated in the first query with $(s, k) = (3, 3)$, and its V_S and V_A are $\{v_2, v_6, v_7, v_8, v_9,\ v_{11}\}$ and $\{v_1, v_4, v_5, v_{10}\}$, respectively. It is not necessary to calculate $U(V_S)$ here, since $U(V_S) = 4$ was derived when solving the first query. According to Eq. (6), when s remains unchanged, $P1$ only needs to appear in the (3,k')-SBs with $k' \geq 4$, which are (3,4)-SB, (3,5)-SB and (3,6)-SB.

- *EE-pruned nodes.* As with the IU-pruned nodes, all EE-pruned nodes will not appear in any (s, k')-SB with $k' \leq k$ for the same reason. Moreover, an EE-pruned node will be pruned again in any (s, k')-SB if $k' - k < p - |V_S| - A(V_S)$, since the social connectivity between V_S and V_A is still too small with respect to k'. Therefore, an EE-pruned node only appears in the (s, k')-SB where

$$k' \geq max\{p - |V_S| - A(V_S) + k, k+1\}.$$

- *Acquaintance-pruned nodes.* An acquaintance-pruned node is included in an (s, k')-SB only if $k' > k$ and $\sum_{v \in M_A} |V_A \cap N_v| \geq (p - |V_S|)(p - |V_S| - k' - 1)$ (i.e., Eq. (5) does not hold to trigger acquaintance pruning). That is, an acquaintance-pruned node only appears in the (s, k')-SB where

$$k' \geq max\{p - |V_S| - 1 - \sum_{v \in M_A} |V_A \cap N_v|/(p - |V_S|), k+1\}.$$

  Note that the value of $\sum_{v \in M_A} |V_A \cap N_v|$ has already been derived in the first query and does not change when k is replaced by k', and exploiting these unchanged parts helps reduce computation when processing SSGQs.

- *Distance-pruned nodes.* In contrast, distance-pruned nodes need to appear and may be expanded in the (s, k')-SB when $k' < k$, since k' represents a tighter acquaintance constraint, and the solutions that trim off the distance-pruned nodes may not be feasible. However, including every distance-pruned node in all (s, k')-SBs in this situation is not necessary. Instead, we employ the distance pruning strategy again to filter out the distance-pruned nodes that never become a better solution in each SB. Specifically, if the solutions generated in the previous queries are feasible under k', the one with the smallest total social distance is kept in the (s, k')-SB, and this total social distance is then employed to update D in distance pruning for filtering. On the other hand, distance-pruned nodes are not included in (s, k')-SBs when $k' \geq k$, because the original solutions that trim off these nodes are still better solutions.

In the above cases, we explored whether a pruned node appears in (s, k')-SB according to its original pruning type. However, taking a distance-pruned node as an example, when it is included in an (s, k')-SB with $k' < k$, it may violate the new tighter interior unfamiliarity condition and be trimmed off by IU pruning. To further reduce the number of nodes, we examine each type of pruned nodes for the other three types of pruning strategies with the corresponding s and k'. These examinations are almost the same as in Section 3.2, except that some parts of the inequalities have already been derived and can be reused directly.

(2) Solution Nodes. It is desirable for the SBs to include solution nodes to facilitate early pruning in the new query. A solution node here can be any node with $|V_S| = p$, e.g., any feasible solution (not necessarily the optimal one). Specifically, any solution node can be selected in an (s, k')-SB if $k' \geq U(V_S)$, since the solution node still satisfies the acquaintance constraint k'. Nevertheless, if there is already another solution node in the (s, k')-SB, we only need to keep the one with the smaller total social distance to facilitate distance pruning afterwards.

(3) Internal Nodes. To effectively minimize the storage overhead, no internal node is included in (s, k')-SBs, since all feasible solutions expanded from an internal node either are the solution nodes in its sub-tree or can be expanded from the pruned nodes in its sub-tree.

Rule 2: Node Indexing for Different s

(1) Pruned nodes.

- *IU-pruned nodes.* No IU-pruned node needs to be included in any (s', k)-SB, since changing the social radius constraint does not increase the connectivity between the existing vertices in V_S. Thus, all the IU-pruned nodes are infeasible with any s'.
- *EE-pruned nodes.* In contrast to the IU-pruned nodes, some EE-pruned nodes may be successfully expanded to generate new sub-trees when $s' > s$, since new candidate attendees may appear in V_A. Therefore, if $s' > s$, it is necessary to derive the corresponding $V_A^{s'}$ (i.e., the candidate attendees within s' hops from the initiator) for an EE-pruned node.[16] We then update the social distance according to different s' to keep track of the status of the pruned node.[17] An (s', k)-SB includes an EE-pruned node

16. The tightest social radius constraint that allows a vertex v to be included as a candidate can be identified from the radius graph extraction procedure, and it is the smallest i such that $d^i_{v,q} < \infty$.

17. The social distance of any vertex in $V_A^{s'}$ for different s' can also be derived from the radius graph extraction procedure, since the social distance of a vertex v for s' is exactly $d^i_{v,q}$ with $i = s'$.

only if its $V_A^{s'}$ is large enough such that $A(V_S) \geq p - |V_S|$, implying that Eq. (3) does not hold and prevents EE pruning.

Example 3. Fig. 2b presents an illustrative example with an EE-pruned node $P5$ to identify the corresponding SBs. $P5$ is generated in the first query with $(s,k) = (3,3)$, and its V_S and V_A are $\{v_3, v_6, v_7, v_8, v_9\}$ and $\{v_1, v_4, v_5, v_{10}, v_{11}\}$, respectively. According to Rule 2-(1), an EE-pruned node is only considered for the (s',k)-SBs with $s' > s$. Since $s_{max} = 4$, where s_{max} is the largest possible s', $P5$ may only stay in the (4,3)-SB. Note that the tightest social radius constraint that allows a vertex v to be included as a candidate can be identified from the radius graph extraction procedure. Therefore, $V_A^4 = V_A^3$. Since $V_A^{s'}$ is unchanged, $A(V_S)$ will remain the same when $s' = 4$, which indicates that Eq. (3) still holds to trim off $P5$ again. By excluding $P5$ from the $(s',3)$-SBs, the node selection rules effectively reduce the processing time of subsequent queries.

Note that, although V_S contains the same set of vertices under different s', the social distances of the vertices in V_S may change and affect the later distance pruning. Therefore, in addition to tracking each node's $V_A^{s'}$ for different s', we update the corresponding V_S for different s', denoted as $V_S^{s'}$. Moreover, $V_S^{s'}$ or $V_A^{s'}$ for the same node under different s' tend to share many common vertices. Therefore, to efficiently maintain $V_S^{s'}$ and $V_A^{s'}$ under different s', we hierarchically save the difference among them. That is, we first save a base node for $V_S^{s'}$ or $V_A^{s'}$ with the smallest s'. When new candidates join or when the social distance of any vertex becomes smaller for a larger s', these new candidates or the difference of social distances will be recorded in a delta node. With the base node and the delta nodes, we can dynamically generate the corresponding $V_S^{s'}$ and $V_A^{s'}$ of the specified s' for further expansion as shown in Example A.3 of Appendix A.1, available in the online supplemental material.

- *Acquaintance-pruned nodes.* Similar to the EE-pruned nodes, some acquaintance-pruned nodes may be expanded into new sub-trees when $s' > s$, since new candidate attendees may appear in V_A. Specifically, an (s',k)-SB includes an acquaintance-pruned node only if its $V_A^{s'}$ is large enough such that

$$\sum_{v \in M_A^{s'}} |V_A^{s'} \cap N_v| \geq (p - |V_S|)(p - |V_S| - k - 1),$$

where $M_A^{s'}$ is the set of $p - |V_S|$ vertices in $V_A^{s'}$ with the largest inner degrees. The above inequality indicates that Eq. (5) does not hold and the node is not pruned.
- *Distance-pruned nodes.* In contrast, most distance-pruned nodes, except those with $V_S^{s'}$ violating the social radius constraint (i.e., $\max_{v \in V_S^{s'}} h_v > s'$, where h_v is the number of hops from the initiator to a vertex v), need to be re-considered when s' changes. The reason is that when $s' > s$, the newly included vertices in $V_A^{s'}$ may create shorter paths to the initiator. Alternately, when $s' < s$, the total social distance of the solution in the previous distance pruning may increase. In either way, the distance pruning condition may not hold, and its pruned nodes need to be included in (s',k)-SB for further examination.

  Here we also create the base node and the delta nodes for a distance-pruned node to compactly maintain its $V_S^{s'}$ and $V_A^{s'}$ for different s' to update the social distance of any vertex, and then use the distance pruning strategy again to include only the updated distance-pruned nodes that can generate a better solution in the (s',k)-SB.

(2) Solution Nodes. In order to facilitate early pruning for the subsequent queries and avoid missing the optimal solution, (s',k)-SB includes the solution nodes that follow the social radius constraint. For each solution node, we update the social distance with any vertex in $V_S^{s'}$, so that it is associated with the correct total social distance. For each (s',k)-SB, we only keep the solution node of the smallest total social distance to reduce the storage overhead.

(3) Internal Nodes. In contrast to the (s,k')-SB, internal nodes in AST play a more important role in the (s',k)-SB, because when $s' > s$, new candidate attendees may join. Therefore, the internal nodes of AST need to be cached for the (s',k)-SBs with $s' > s$, so that new candidates can be added to the existing internal nodes without generating them all over again. Similar to the pruned nodes, we maintain $V_S^{s'}$ and $V_A^{s'}$ of each internal node for different s' using the base node and the delta nodes, so that we can dynamically generate the corresponding $V_S^{s'}$ and $V_A^{s'}$ of the specified s' for further expansion.

Although the node indexes for different k and different s have been presented in Rule 1 and Rule 2, respectively, when considering both s and k, carefully combining the rules for k and for s can further reduce the number of nodes to include in SBs. Moreover, when updating AST and SB during the query processing for a new SGQ, the number of nodes included in SBs can be minimized by considering the s of all historical queries simultaneously. Therefore, Appendix C.1, available in the online supplemental material, first explores the generalized rule, and the updating procedure of AST and SB is then detailed in Appendix C.2, available in the online supplemental material.

Although we only process a small portion of nodes in AST (i.e., the nodes indexed by the SB), the following theorem proves that the solution is still ensured to be optimal.

Theorem 3. *Processing the nodes in the (s',k')-SB obtains the optimal solution to the SSGQ.*

Proof. We prove the theorem in Appendix D.4, available in the online supplemental material. □

In Appendix D.5, available in the online supplemental material, we further prove that all the nodes included in the SB are feasible in Theorem 7. Additionally, we show that the nodes included in the SB are sufficient and necessary in Corollary 1.

6 Extensions in Temporal Dimension

The proposed AST and SB can be extended to support a sequence of STGQs by considering the temporal dimension. Specifically, it is only necessary to record T_S (i.e., the common available time interval of all vertices in V_S) for the

nodes in AST and attach the available time to the vertices in V_A. As introduced in Section 3.3, when processing an STGQ, instead of considering every interval from t to $t+m-1$ for each time slot t, our algorithm leverages the pivot time slot im to effectively reduce the search space. Therefore, the amount of ASTs can also be reduced from the number of time slots to the number of pivot time slots.

Although the node selection rules in Section 5 were derived for a sequence of SGQs, these rules can be directly applied to a sequence of STGQs with no loss of solution optimality. The reason is that, based on the proof of Theorem 3, all the excluded nodes still cannot generate the optimal solution to a subsequent STGQ with the same s' and k' as with the subsequent SGQ, since adding the availability constraint m in the STGQ does not loosen the social constraints or decrease the total social distance of these nodes.

By considering candidates' availability in the temporal dimension, the size of each SB can be effectively reduced. For example, if a node is pruned due to the lack of temporal extensibility (i.e., $X(V_S) < 0$), it is not required in any SB. A lack of temporal extensibility implies a lack of consecutive time slots for vertices in the V_S of the node. The s' and k' of a new query do not affect schedules, and the consecutive time slots for vertices in the V_S remain insufficient. Hence, this node will again be pruned in the new query.

Similarly, if a node is pruned by availability pruning, it does not need to be considered under most situations, since the schedules of the vertices in V_A are not affected. Note that one exception is that when s' of the new query becomes larger, additional vertices may appear in V_A, and these vertices could form a feasible solution with V_S if they have enough time slots in common. Therefore, when exploiting the temporal availability to further reduce the size of each SB, we only discard the nodes that do not have enough new vertices in V_A to ensure the solution optimality while reducing the processing time.

7 Experimental Results

In this section, we evaluate the performance and analyze the solution quality of the proposed algorithms. First, we describe the experiment setup in Section 7.1. We perform a series of sensitivity tests to study the impact of query parameters with real datasets and evaluate the performance of Algorithms SGSelect and STGSelect in Section 7.2. In Section 7.3, we further analyze experimental results of SSGQ with the proposed caching mechanisms.

7.1 Experiment Setup

To evaluate the performance and analyze the solution quality of the proposed algorithms, we conduct various experiments using real datasets. The existing approaches (e.g., [10], [11], [12]) cannot be applied to solve SGQ and STGQ because their problem formulations do not include the temporal dimension and social constraints s and k. Therefore, we compare the proposed algorithms with three other approaches: SGBasic (i.e., enumerating all possible candidate groups), KNN (i.e., selecting the $p-1$ people with the smallest social distances to the initiator), and DKS [37] (i.e., choosing the candidate group out of all possible ones that maximizes the number of edges within the group). Note that DKS is the core of the algorithms in the aforementioned works [10], [11], [12] and correlated to [6], [38], [39].

The experiment includes two datasets: (1) the `Coauthor` dataset [40] with 16,726 people and 117,082 days of schedule, and (2) the `YouTube` dataset [31] with 1,134,890 people and 7,944,230 days of schedule. For the temporal information, because there is no public dataset on real user schedules (probably due to privacy concerns), we can only collect the real user schedules from a user study with 194 participants who shared their Google Calendar to us for research purpose. The collected `Google Calendar` dataset from the user study includes 6,790 days of real schedules. The temporal information in the above `Coauthor` and `YouTube` datasets are randomly selected from the 6,790 days accordingly. We randomly select 5 weekdays and 2 weekend schedules to form a 7-day schedule for each vertex in the social networks. To evaluate SSGQ thoroughly, we conduct experiments with various parameter settings for SSGQ. Specifically, the parameters are sequentially increased or decreased with different rates. Moreover, we also conduct experiments with randomly parameter adjustment. The algorithms are implemented on an HP DL580 server with four Intel E7-4870 2.4 GHz CPUs.

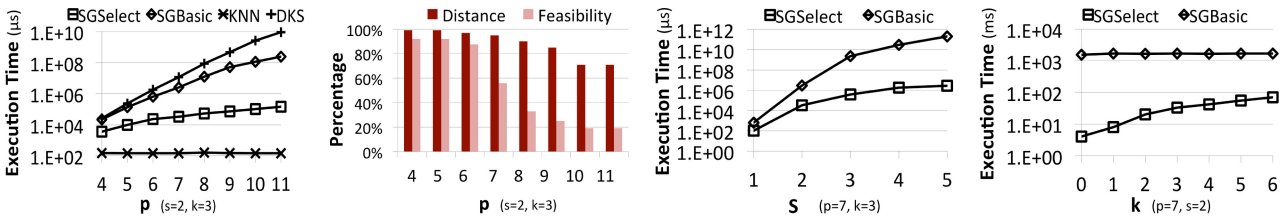
7.2 Performance Analysis of SGQ and STGQ

Analysis of SGQ. We first compare the running time of SGSelect against SGBasic, KNN, and DKS with different group size p. Fig. 3a presents the results with $s=2$ and $k=3$. The trend in other parameter settings, such as $s=1$, is similar. Results show that SGSelect outperforms SGBasic and DKS more significantly as p grows, since SGBasic and DKS need to examine numerous candidate groups, and the processing effort of each group increases with p. In contrast, SGSelect effectively prunes the solution space with the proposed access ordering and pruning strategies.

Although KNN is fastest in Fig. 3a, Fig. 3b shows that many solutions returned by KNN are infeasible to SGQ. Specifically, the *feasibility* ratio shows the percentage of feasible groups returned by KNN drops quickly as p grows, since the candidates close to the initiator do not necessarily know each other. In addition, Fig. 3b compares the total social distances of KNN and SGSelect. The *distance* ratio represents the total social distance returned by KNN divided by that returned by SGSelect. Note that the solution of KNN, with relaxed social constraints, can be regarded as a lower bound on the total social distance of SGQ. Fig. 3b indicates that the ratio remains above 70 percent even for large p.

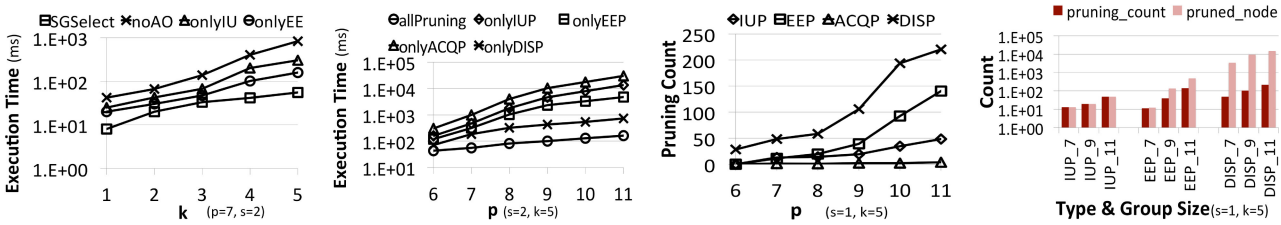
Fig. 3c presents the results with different social radius constraints. As s rises, the number of candidate vertices (i.e., friends within s hops) increases quickly, and consequently, the running time escalates. When s changes from 2 to 3, the running time of SGBasic becomes nearly 1,000 times greater while that of SGSelect only increases 11-fold, indicating the effectiveness of the proposed strategies. We also compare these two approaches under different acquaintance constraints. As shown in Fig. 3d, the running time of SGBasic changes only slightly for different k, while the running time of SGSelect is reduced for a smaller k, since the IU pruning, EE pruning and acquaintance pruning become more effective.

Detailed Analysis on Proposed Strategies. In the following, we first investigate the effectiveness of the access ordering strategy. Fig. 4a shows that using only interior unfamiliarity (see `onlyIU`) or exterior expansibility (see `onlyEE`) reduces the running time, and the proposed access ordering strategy that combines them yields the greatest improvement. Next, Figs. 4b, 4c, and 4d analyze the pruning power of



(a) Comparison of running time with different p.
(b) Solution quality analysis of KNN.
(c) Comparison of running time with different s.
(d) Comparison of running time with different k.

Fig. 3. Experimental results of SGQ (with the `Coauthor` dataset).



(a) Comparison of running time with different access ordering strategies used.
(b) Comparison of running time with different pruning strategies used.
(c) Average pruning counts of different strategies.
(d) Comparison between the pruning count and the pruned node count. (The postfix represents the group size p.)

Fig. 4. Analysis on pruning ability of proposed strategies (with the `Coauthor` dataset).

acquaintance pruning, distance pruning, IU pruning (i.e., the prunings contributed by Lemma 1), and EE pruning (i.e., the prunings contributed by Lemma 2). Fig. 4b compares the running time of SGSelect using different pruning strategies. Fig. 4c then analyzes the effectiveness of these pruning strategies by comparing their pruning counts. Distance pruning is the most effective one due to the help of the access ordering strategy, i.e., the first feasible solution with a small total social distance returned by access ordering can be exploited to facilitate effective distance pruning. On the other hand, the pruning count of EE pruning exceeds that of IU pruning as p increases, because under the same k, the number of required edges inside a size-p feasible group (i.e., $p(p-k-1)/2$) increases as p grows, making it more difficult to satisfy the external expansibility condition.

Finally, Fig. 4d compares the pruned node count and the pruning count with different strategies. Each pruning can remove multiple nodes in the search tree, and the pruned node count thereby can be larger than the pruning count. The pruning ratio (i.e., the pruning count to pruned node count) of distance pruning reaches $1:90$. When a pruning happens in a position closer to the root of the tree, it prunes off a larger branch and the number of pruned nodes increases. Therefore, we further investigate the position of pruning in different strategies. Table 2 shows the percentage of prunings that occur when $|V_S| \leq \lfloor \frac{p}{2} \rfloor$, where a smaller $|V_S|$ indicates proximity to the root. The IU pruning usually occurs farther from the root because the pruning requires the LHS of Eq. (1) to exceed the RHS, and the value of LHS tends to increase as $|V_S|$ becomes larger. Nevertheless, the IU pruning still plays an important role in SGSelect because it prunes off the infeasible V_S and ensures that the final solution satisfies the acquaintance constraint.

TABLE 2
The Percentage of Prunings Located Near the Root of the Dendrogram

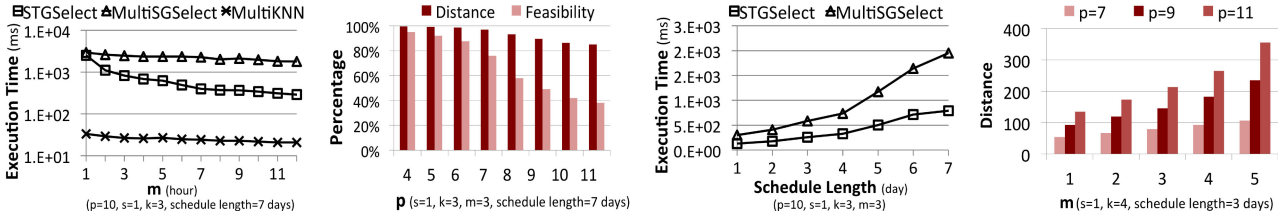
Group Size	IUP	EEP	DISP
p=7	0%	44%	61%
p=9	0%	52%	60%
p=11	0%	61%	64%

Analysis on Temporal Dimension. To evaluate the performance on STGQ, we compare STGSelect with the following algorithms: MultiSGSelect, MultiKNN, and MultiDKS, i.e., sequentially considering each candidate activity period and solving the corresponding SGQ problem using SGSelect, KNN, and DKS, respectively. Fig. 5a compares the running time of these algorithms under different activity lengths m. Note that the running time of MultiDKS is more than 7 hours and thereby not plotted in this figure. The results show that STGSelect outperforms MultiSGSelect, especially for a larger m, due to a significantly smaller number of pivot time slots examined in STGSelect. For KNN, though MultiKNN is the fastest, it is not able to guarantee a feasible solution. Fig. 5b shows that the percentage of feasible groups returned by MultiKNN drops quickly as p grows, and the *distance* ratio remains above 85 percent, better than the simpler case of 70 percent in Fig. 3b.

Fig. 5c further presents the running time of STGSelect and MultiSGSelect with varied users' schedule lengths, and STGSelect consistently outperforms MultiSGSelect. Fig. 5d further analyzes the solution quality with various m. For each p, the total social distance steadily increases as m grows, because longer activity periods may necessitate the candidates with larger social distances. To show the scalability of proposed algorithms and provide further analysis on solution quality, we also evaluate them in a larger `YouTube` dataset, and the results are provided in Appendix E.3, available in the online supplemental material.

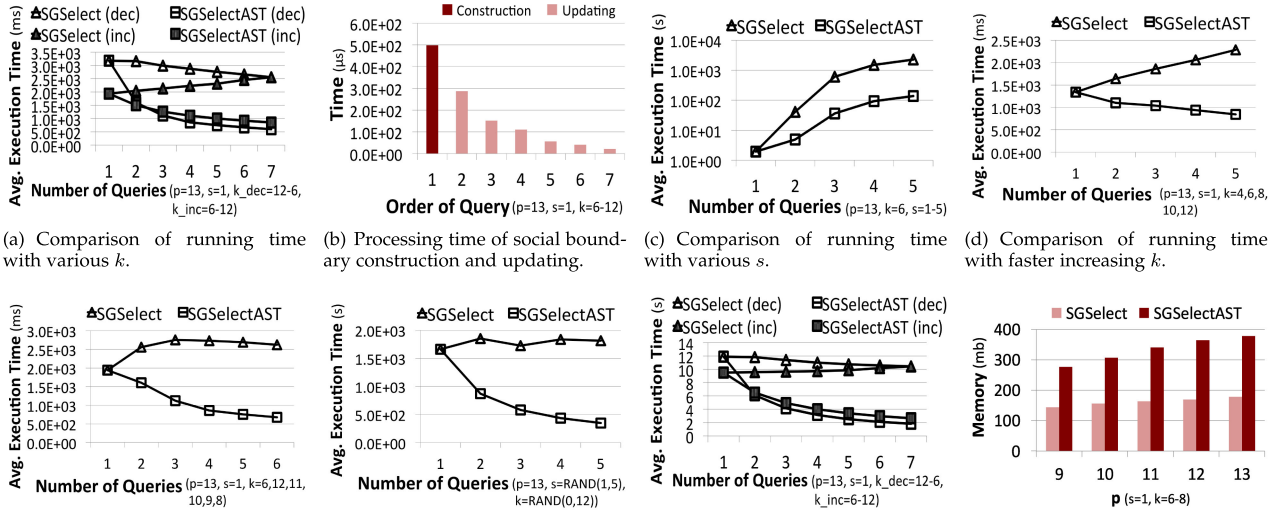
7.3 Performance Analysis of SSGQ

In the following, we compare the average query time for processing SSGQs with and without the proposed index structure for subsequent queries (i.e., SGSelectAST and



(a) Comparison of running time with different m.
(b) Solution quality analysis of KNN.
(c) Comparison of running time with different schedule lengths.
(d) Comparison of solution quality with different m.

Fig. 5. Experimental results of STGQ (with the Coauthor dataset).



(a) Comparison of running time with various k.
(b) Processing time of social boundary construction and updating.
(c) Comparison of running time with various s.
(d) Comparison of running time with faster increasing k.
(e) Comparison of running time with dynamic k.
(f) Comparison of running time with dynamic s and k.
(g) Comparison of running time with various k on the YouTube dataset.
(h) Comparison of memory consumption with various p.

Fig. 6. Experimental results of SSGQ (with the Coauthor dataset, except for Fig. 6g).

SGSelect). Fig. 6a shows the results under varied numbers of consecutive queries (in x-axis). The first query starts with $k = 6$ and $k = 12$ for experiments with increasing k and decreasing k, respectively. For example, the result for $x = 3$ with a decreasing k represents the average query processing time of three consecutive queries for $k = 12$, 11 and 10. The results manifest that the average query processing time of SGSelectAST outperforms SGSelect because the proposed AST and SB effectively avoid exploring duplicate search space. For a decreasing k, the later queries are subject to smaller k (i.e., stricter social constraints), and larger solution space is more inclined to be pruned. This explains the decrease of average query processing time for both SGSelect and SGSelectAST. Similarly, for an increasing k, we can infer that the average query processing time tends to become longer as the number of consecutive queries increases. Surprisingly, the average running time of SGSelectAST still decreases consistently, and its improvement over SGSelect becomes more significant as the number of queries grows. Compared with SGSelect, the query processing time of SGSelectAST is only 50 percent for two consecutive queries and 25 percent for five consecutive queries. As expected, AST preserves more information to effectively trim the solution space required to be explored.

Fig. 6b evaluates the overhead of constructing and updating SBs. The result manifests that the first query incurs the largest overhead since all the SBs need to be carefully examined and created. The overhead in later updates decreases quickly for an increasing k, because the number of pruned nodes becomes smaller. Note that the total overhead only contributes less than 0.1 percent of the query processing time of the SSGQs and hence is quite ignorable. Moreover, the construction and update of SBs may be performed offline before a new query is issued.

Fig. 6c compares the running time under different s. When s grows from 1 to 5, the average processing time for SGSelect is boosted by 1162 times. In contrast, SGSelectAST only increases by 71 times, indicating that the proposed AST and SB can alleviate the growth of computational overhead significantly. Considering that the initiator may not always tighten or loosen the social constraints by one at a time, we evaluate SGSelectAST by increasing k in a faster rate in Fig. 6d. We further consider a more dynamic case in Fig. 6e, where the initiator first overly increases k from 6 to 12, and then slowly decreases it to 8. In Fig. 6f, the s and k of each query are randomized to simulate a case of random parameter tuning. In all the above cases, SGSelectAST always outperforms SGSelect. We also compare the algorithms in the large YouTube dataset with an increasing and decreasing k in Fig. 6g. The average query processing time of SGSelect becomes smaller with a decreasing k but becomes larger with an increasing k. In contrast, the average query processing time of SGSelectAST steadily becomes smaller in both cases.

Fig. 6h shows the peak memory consumption of SGSelect and SGSelectAST under different p in the Coauthor dataset. For each p, there are three queries with $k = 6$, 7 and 8. For example, when $p = 13$, the three $SGQ(p, s, k)$ queries issued are $SGQ(13, 1, 6)$, $SGQ(13, 1, 7)$, and $SGQ(13, 1, 8)$. SGSelect

processes the three queries separately as three SGQs, whereas SGSelectAST builds AST according to previous queries to speed up the subsequent queries. Fig. 6h manifests that the memory consumption of SGSelectAST becomes larger when AST grows with p. However, the social boundary and the derived node selection rules effectively avoid storing redundant intermediate solutions, and the cached intermediate solutions effectively reduce the computation time as shown in Fig. 6a. Compared with SGSelect, the query processing time of SGSelectAST is only 50 percent for two consecutive queries and 25 percent for five consecutive queries. Most importantly, AST only needs to be cached for a short period of time during the activity planning.

8 Conclusion

To the best of our knowledge, there is no existing work in the literature that addresses the issues of automatic activity planning based on both the social and temporal relationships of an initiator and attendees, especially for a sequence of queries with different parameters from users to iteratively find more desired solutions. In this study, we first define two useful queries, namely, SGQ and STGQ, to obtain the optimal set of attendees and suitable activity time. We show that these problems are NP-hard and inapproximable within any ratio. We then devise two algorithms, SGSelect and STGSelect, to find optimal solutions in reasonable time with effective query processing strategies, including access ordering, distance pruning, acquaintance pruning, pivot time slots, and availability pruning are explored to prune redundant search space for efficiency. To support and accelerate subsequent social group queries (SSGQs), we further propose AST and SB. By effectively caching and indexing the intermediate solutions of previous queries, data lookup needed for SSGQs is facilitated. Experimental results show that, with AST and SB, the running time of SSGQs is reduced by 50-75 percent.

Acknowledgments

This work was supported in part by US NSF through grants IIS-1717084 and SMA-1360205, and by the Ministry of Science and Technology in Taiwan through grants 105-2218-E-002-035, 105-2218-E-007-032-MY2, 106-3114-E-468-001, 106-2218-E-002-044, 107-2636-E-007-003, and 107-2636-E-011-002.

References

[1] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 44–54.

[2] M. Yuan, L. Chen, P. S. Yu, and T. Yu, "Protecting sensitive labels in social network data anonymization," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 3, pp. 633–647, Mar. 2013.

[3] J. Chen, W. Geyer, C. Dugan, M. J. Muller, and I. Guy, "Make new friends, but keep the old: Recommending people on social networking sites," in *Proc. 27th ACM SIGCHI Int. Conf. Human Factors Comput. Syst.*, 2009, pp. 201–210.

[4] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. M. A. Patwary, "Fast maximum clique algorithms for large graphs," in *Proc. 23rd Int. Conf. World Wide Web*, 2014, pp. 365–366.

[5] J. Xiang, C. Guo, and A. Aboulnaga, "Scalable maximum clique computation using MapReduce," in *Proc. 29th Int. Conf. Data Eng.*, 2013, pp. 74–85.

[6] C. E. Tsourakakis, "The k-clique densest subgraph problem," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 1122–1132.

[7] D. Berlowitz, S. Cohen, and B. Kimelfeld, "Efficient enumeration of maximal K-plexes," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2015, pp. 431–444.

[8] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2014, pp. 1311–1322.

[9] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 467–476.

[10] A. An, M. Kargar, and M. ZiHayat, "Finding affordable and collaborative teams from a network of experts," in *Proc. SIAM Int. Conf. Data Mining*, 2013, pp. 587–595.

[11] S. S. Rangapuram, T. Bühler, and M. Hein, "Towards realistic team formation in social networks based on densest subgraphs," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1077–1088.

[12] K. Li, W. Lu, S. Bhagat, L. V. S. Lakshmanan, and C. Yu, "On social event organization," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1206–1215.

[13] A. Maunz, C. Helma, and S. Kramer, "Efficient mining for structurally diverse subgraph patterns in large molecular databases," *Mach. Learn.*, vol. 83, no. 2, pp. 193–218, 2011.

[14] Z. Yang, A. W. Fu, and R. Liu, "Diversified top-k subgraph querying in a large graph," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2016, pp. 1167–1182.

[15] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1965–1974.

[16] B. He, Y. Ding, J. Tang, V. Reguramalingam, and J. Bollen, "Mining diversity subgraph in multidisciplinary scientific collaboration networks: A meso perspective," *J. Informetrics*, vol. 7, no. 1, pp. 117–128, 2013.

[17] S. Janson and M. J. Luczak, "A simple solution to the *k*-core problem," *Random Struct. Algorithms*, vol. 30, no. 1–2, pp. 50–62, 2007.

[18] V. Batagelj and M. Zaversnik, "Fast algorithms for determining (generalized) core groups in social networks," *Adv. Data Anal. Classification*, vol. 5, no. 2, pp. 129–145, 2011.

[19] S. Shahinpour and S. Butenko, "Algorithms for the maximum K-club problem in graphs," *J. Combinatorial Optimization*, vol. 26, no. 3, pp. 520–554, 2013.

[20] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum *K*-plex problem," *Operations Res.*, vol. 59, no. 1, pp. 133–142, 2011.

[21] B. McClosky and I. V. Hicks, "Combinatorial algorithms for the maximum *K*-plex problem," *J. Combinatorial Optimization*, vol. 23, no. 1, pp. 29–49, 2012.

[22] H. Moser, R. Niedermeier, and M. Sorge, "Algorithms and experiments for clique relaxations-finding maximum S-plexes," in *Proc. 8th Int. Symp. Exp. Algorithms*, 2009, pp. 233–244.

[23] B. Wu and X. Pei, "A parallel algorithm for enumerating all the maximal *k*-plexes," in *Proc. 11th PAKDD Int. Workshop Emerging Technol. Knowl. Discovery Data Mining*, 2007, pp. 476–483.

[24] R. Mokken, "Cliques, clubs and clans," *Quality Quantity*, vol. 13, pp. 161–173, 1979.

[25] C. Li and M. Shan, "Composing activity groups in social networks," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manag.*, 2012, pp. 2375–2378.

[26] T. T. Vu, D. Song, A. Willis, S. N. Tran, and J. Li, "Improving search personalisation with dynamic group formation," in *Proc. 37th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2014, pp. 951–954.

[27] S. B. Roy, L. V. S. Lakshmanan, and R. Liu, "From group recommendations to group formation," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2015, pp. 1603–1616.

[28] D. Yang, C. Shen, W. Lee, and M. Chen, "On socio-spatial group query for location-based social networks," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 949–957.

[29] Y. Li, D. Wu, J. Xu, B. Choi, and W. Su, "Spatial-aware interest group queries in location-based social networks," *Data Knowl. Eng.*, vol. 92, pp. 20–38, 2014.

[30] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 939–948.

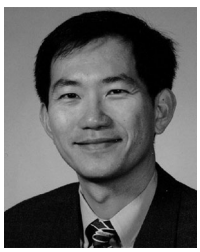
[31] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. 12th IEEE ICDM Int. Conf. Data Mining*, 2012, pp. 745–754.

[32] J. J. McAuley and J. Leskovec, "Discovering social circles in ego networks," *ACM Trans. Knowl. Discovery Data*, vol. 8, no. 1, pp. 4:1–4:28, 2014.
[33] E. Kanoulas, B. Carterette, P. D. Clough, and M. Sanderson, "Evaluating multi-query sessions," in *Proc. 34th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2011, pp. 1053–1062.
[34] M. Sloan and J. Wang, "Dynamical information retrieval modelling: A portfolio-armed bandit machine approach," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 603–604.
[35] C. Li, P. Resnick, and Q. Mei, "Multiple queries as bandit arms," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manag.*, 2016, pp. 1089–1098.
[36] S. Milgram, "The small world problem," *Psychology Today*, vol. 2, pp. 60–67, 1967.
[37] U. Feige, G. Kortsarz, and D. Peleg, "The dense *K*-subgraph problem," *Algorithmica*, vol. 29, no. 3, pp. 410–421, 2001.
[38] L. Qin, R. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 965–974.
[39] A. Epasto, S. Lattanzi, and M. Sozio, "Efficient densest subgraph computation in evolving graphs," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 300–310.
[40] M. E. J. Newman, "The structure of scientific collaboration networks," *Proc. Nat. Academy Sci. United States America*, vol. 98, no. 2, pp. 404–409, 2001.
[41] M. Kargar and A. An, "Discovering top-k teams of experts with/without a leader in social networks," in *Proc. 20th ACM Conf. Inf. Knowl. Manag.*, 2011, pp. 985–994.
[42] M. Juang, C. Huang, and J. Huang, "Efficient algorithms for team formation with a leader in social networks," *J. Supercomputing*, vol. 66, no. 2, pp. 721–737, 2013.
[43] Z. Lv, J. Huang, Y. Zhou, H. Sun, and X. Jia, "Grouped team formation in social networks," in *Proc. 18th Asia-Pacific Web Conf. Web Technol. Appl.*, 2016, pp. 398–401.
[44] M. Kargar, A. An, and M. Zihayat, "Efficient bi-objective team formation in social networks," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2012, pp. 483–498.
[45] A. Majumder, S. Datta, and K. V. M. Naidu, "Capacitated team formation problem on social networks," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1005–1013.
[46] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T. Chua, "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2017, pp. 335–344.
[47] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 173–182.
[48] Y. Gu, B. Zhao, D. Hardtke, and Y. Sun, "Learning global term weights for content-based recommender systems," in *Proc. 25th Int. Conf. World Wide Web*, 2016, pp. 391–400.
[49] Z. Lu, Z. Dou, J. Lian, X. Xie, and Q. Yang, "Content-based collaborative filtering for news topic recommendation," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 217–223.
[50] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proc. 8th IEEE Int. Conf. Data Mining*, 2008, pp. 263–272.
[51] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 426–434.
[52] R. M. Karp, "Reducibility among combinatorial problems," in *Proc. Symp. Complexity Comput. Computations*, 1972, pp. 85–103.
[53] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proc. 4th SIAM Int. Conf. Data Mining*, 2004, pp. 442–446.
[54] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt, "Recommendations to boost content spread in social networks," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 529–538.
[55] K. Mouratidis, J. Li, Y. Tang, and N. Mamoulis, "Joint search by social and spatial proximity," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3, pp. 781–793, Mar. 2015.

Yi-Ling Chen received the BS degree from the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, and the PhD degree from the Department of Electrical Engineering, National Taiwan University, Taipei. She is now an assistant professor with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology. Her research interests include social network analysis, data mining, and pervasive computing.

De-Nian Yang received the BS and PhD degrees from the Department of Electrical Engineering, National Taiwan University. He is now a research fellow with the Institute of Information Science, Academia Sinica, Taiwan. His research interests include mobile social networks and mobile multimedia networking. He received the K.-T. Li Distinguished Young Scholar Award in the ACM Taipei/Taiwan Chapter, the Research Exploration Award from the Pan Wen Yuan Foundation, and Project for Excellent Junior Research Investigators in NSC, Taiwan. He is a senior member of the IEEE and a member of the ACM.

Chih-Ya Shen received the BS and MS degrees from the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, and the PhD degree from the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. He is now an assistant professor with the Computer Science Department, National Tsing Hua University. His research interests include mobile computing and query processing.



Wang-Chien Lee received the BS degree from the National Chiao Tung University, Hsinchu, Taiwan, the MS degree from Indiana University, Bloomington, and the PhD degree from the Ohio State University, Columbus. He is an associate professor of computer science and engineering at the Pennsylvania State University, University Park. He is particularly interested in developing data management techniques for supporting complex services in a wide spectrum of networking and mobile environments. He is a member of the IEEE.

Ming-Syan Chen received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, and the MS and PhD degrees in computer, information and control engineering from the University of Michigan, Ann Arbor. He is now the dean of the College of EECS and a distinguished professor jointly appointed by the EE Department, the CSIE Department, and the Graduate Institute of Communication Engineering (GICE) at National Taiwan University. His research interests include databases, data mining, and cloud computing. He is a fellow of the ACM and a fellow of the IEEE.