

國立臺灣大學電機資訊學院電機工程學系

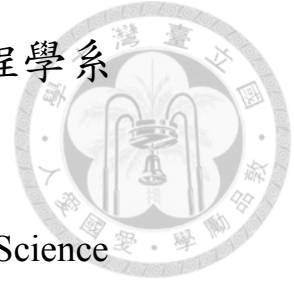
博士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Doctoral Dissertation



於社群網路中之高效能鏈結預測與群組查詢

Efficient Link Prediction and Group Query

in Social Networks

陳怡伶

Yi-Ling Chen

指導教授：陳銘憲博士

Advisor: Ming-Syan Chen, Ph.D.

中華民國 106 年 1 月

January, 2017



國立臺灣大學博士學位論文  
口試委員會審定書

於社群網路中之高效能鏈結預測與群組查詢  
Efficient Link Prediction and Group Query in Social  
Networks

本論文係陳怡伶君 (F96921038) 在國立臺灣大學電機工程學系完成之博士學位論文，於民國 106 年 1 月 25 日承下列考試委員審查通過及口試及格，特此證明。

口試委員：

陳紹堯

(簽名)

(指導教授)

陳衣錫

曾新修

楊得年

沈之滢

系主任

劉志文

(簽名)





## 誌謝

在博士班的旅程中，首先特別要感謝我的指導教授陳銘憲老師，一路上給予我許多幫助，用幽默的言語鼓勵我，並提供了許多中肯的建議。陳老師精準的眼光和判斷，在諸多關鍵時刻，為我指引了明確的方向。而陳老師做研究的積極態度、為人處世的剛柔並濟，都是我學習的典範。陳老師不僅是我的論文指導教授，更是我人生的良師。

在過去幾年間，有幸能與李旺謙老師、楊得年老師、沈之涯老師、鄭文皇老師、游創文老師一起合作發表研究成果，每位老師的研究主題和風格雖不盡相同，但對於學術的熱忱都是一致的。在每篇論文從無到有的過程中，非常感謝諸位老師的無私相授，讓我學習到許多做研究的方法和態度。而在美國伊利諾大學芝加哥分校訪問的一年期間，很幸運地能與 Prof. Philip S. Yu 一起合作，俞教授豐富的學術涵養與對社會時事的關心，都讓我非常敬佩且獲益良多。

在博士論文計畫審查階段，非常感謝林守德老師和于天立老師指出許多可以進一步深入研究的要點，讓論文的內容更加豐富。而在博士學位考試階段，更要感謝陳良弼老師、曾新穆老師、楊得年老師、沈之涯老師費心審閱並給予指教，諸位老師的建議是讓本篇論文更加進步的關鍵。

此外，感謝 NetDB 實驗室的每一位成員給予我的陪伴、鼓勵和協助，你們在我讀博士班的過程中帶來了許多歡笑和溫暖。在學術上因為與你們一起鑽研而進步，而生活也因為與你們一起分享而精彩，這份一起努力打拼的革命情感我永遠難忘。也謝謝一路上幫我加油打氣的許多朋友，特別是遠在美國的信好學姊和雅婷學姊，以及從高中畢業後未曾間斷聯繫的瑋玲、秀帆、珮珊、如君，你們對我的關心和幫助我始終感銘於心。最後，我要向我的家人致上最誠摯的謝意，你們的愛是最堅強的後盾，支持著我完成博士班這個重要里程碑。

雖然即將結束學生身份，但在人生各個方面，學習的旅程依然持續。我會帶著大家給予的支持和鼓勵，心懷感激地繼續努力，把博士班所獲得的能力和知識做出更好的發揮。每一天都是一個新的練習，願能在每次練習中繼續成長蛻變，往期許的樣貌前進。





## 摘要

隨著社群網站的普及與蓬勃發展，許多推薦系統開始利用社群網路中的資訊來提供對使用者有助益的建議，諸多與社群網路分析相關的研究也隨之展開。近來社群網路的規模快速增長，導致推薦系統的運算成本顯著增加。對於要處理社群網路資料以提供建議的推薦系統而言，社群網路的複雜性和巨大規模帶來了沈重的運算負擔。因此，在本論文中，我們針對社群網路中的三個重要推薦問題進行研究，並致力於提升它們的運算效率。

首先，我們聚焦於兩個使用者間的關係來研究大型網路中的鏈結預測問題。在進行鏈結預測時，許多特徵值需要被計算並且整合以便進行推薦，而這些運算成本會隨網路規模成長而快速增加。先前部分關於網路處理之研究嘗試透過稀疏化縮小網路規模，以便降低運算成本。然而，重要的資訊可能在稀疏化過程中被移除，因而導致預測準確率大幅下降。為了解決這個問題，我們提出了一個名為 DEDS 的架構，它能建立具有高度準確率的整體分類器，同時能降低預測所需的時間。DEDS 包含了多樣的稀疏化方法，而這些方法是為了保存網路中的不同特性所設計的。因此，DEDS 能夠產生出具有顯著結構差異性的稀疏網路，並且增加整體分類器的多樣性，從而提升預測效果。

接著我們將討論範疇從兩個使用者間的關係擴展到一群使用者之間的關係，並研究社群群組查詢問題與活動規劃的相關應用。考慮所有使用者間的社交鏈結以推薦一群相互認識的活動參加者，這是一個非確定性多項式時間複雜性類 (NP-hard) 問題。除了找一群相互熟識的活動參加者外，選定一個所有參加者皆有空的時間也是活動規劃的關鍵要素。因此，我們還需要額外考慮使用者的有空時間，而社交連結複雜性和使用者行程的多樣性使得這個問題變得更加困難。在本論文中，我們提出社群時域群組查詢 (Social-Temporal Group Query) 來找到合適的活動時間與一群具有最小社群距離總和的參加者。我們並設計了兩個演算法，分別是 SGSelect 和 STGSelect，它們包含了多種有效的修剪策略 (pruning strategy) 來大幅減少執行時間。實驗結果顯示，我們設計的演算法比起基準方法有明顯的效率提升。我們也進行

了使用者研究，將所提出的演算法與人工活動規劃進行比較。研究結果顯示所提出的演算法能取得較高品質的答案，且需要的規劃工作量較少，因此能夠增加使用者發起活動的意願。

最後，我們研究了連續群組查詢問題，以便有效支援一連串的推薦任務。在規劃活動時，使用者通常不容易將所有條件一次設定完備並找到完美的活動參加者群組與活動時間。幸好透過前述的社群時域群組查詢，使用者可以很容易地調整參數並且獲得其他推薦結果以供選擇。有鑒於使用者可能反覆變動參數以便微調結果，我們進一步提出連續社群群組查詢（Consecutive Social Group Query）來支援此需求。考量到利用先前查詢的中間解將能增進後續查詢的效率，我們設計了一個名為累積搜尋樹（Accumulative Search Tree）的樹狀結構，用緊緻形式暫存歷史查詢的中間解以供重複利用。為了提升查找效率，我們進一步設計了一個名為社群邊界（Social Boundary）的索引結構，在處理特定參數的連續社群群組查詢時，可快速地取得所需使用的中間解。根據實驗結果顯示，透過所設計的暫存機制，連續查詢所花的處理時間將可進一步被顯著降低。

關鍵字：演算法設計與分析、社群網路、鏈結預測、網路稀疏化、查詢處理、索引結構



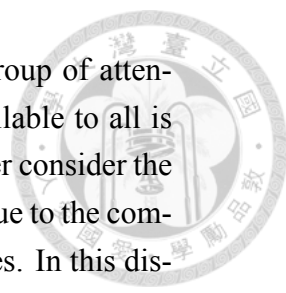


# Abstract

As the development and popularization of social networking websites, many recommendation systems tend to leverage the information in social networks to provide helpful suggestions for users, and a great deal of research studies on social network analysis are thereby motivated. Recently, the sizes of social networks have been increasing rapidly, and this growth results in a significant increase in the computational cost of the sophisticated recommendations. The huge size and complexity of social networks create a considerable burden for recommendation systems while processing the information from social networks to provide suggestions. Therefore, in this dissertation, we study three important recommendation problems in social networks and aim to improve their efficiency.

First, we focus on the relationship between two users and study the link prediction problem in large-scale networks. During the link prediction, numerous feature values need to be calculated and then combined to make recommendations, and the computational cost grows quickly as the network size becomes larger. Some previous studies involving network processing attempt to lower the computational cost by reducing the network size via sparsification. However, sparsification might remove important information and hurt the prediction accuracy. To address this issue, we propose a framework called *Diverse Ensemble of Drastic Sparsification (DEDS)*, which constructs ensemble classifiers with good accuracy while keeping the prediction time short. DEDS includes various sparsification methods that are designed to preserve different measures of a network. Therefore, DEDS can generate sparsified networks with significant structural differences and increase the diversity of the ensemble classifier, which is key to improving prediction performance.

Second, we extend the scope from the relationship between two users to the relationship among a group of users, and study the social group query problem with its applications in activity planning. Considering social links among all users to recommend a mutually acquainted group of attendees for



an activity is an NP-hard problem. In addition to finding a group of attendees familiar with each other, selecting an activity period available to all is also essential for activity planning. Therefore, we need to further consider the available time of users, which makes the problem even harder due to the complexity of social connectivity and the diversity of user schedules. In this dissertation, we propose the *Social-Temporal Group Query (STGQ)* to find suitable time and attendees with minimum total social distance. We design two algorithms, *SGSelect* and *STGSelect*, which include various effective pruning strategies to substantially reduce running time. Experimental results indicate that *SGSelect* and *STGSelect* are significantly more efficient than baseline approaches. We also conduct a user study to compare the proposed approach with manual activity coordination. The results show that our approach obtains higher quality solutions with less coordination effort, thereby increasing users' willingness to organize activities.

Finally, we study the consecutive group query problem to support a sequence of recommendations. When planning an activity, it is difficult for a user to specify all the conditions right at once to find the perfect group of attendees and time. Fortunately, with the aforementioned social-temporal group query, it is easy for the user to tune the parameters to find alternative recommendations. As users may iteratively adjust query parameters to fine tune the results, we further propose *Consecutive Social Group Query (CSGQ)* to support such needs. Envisaging that exploiting the intermediate solutions of previous queries may improve processing of the succeeding queries, we design a new tree 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 a new index structure, called *Social Boundary*, which effectively indexes the intermediate solutions required for processing each CSGQ with specified parameters. According to the experimental results, with the caching mechanisms, processing time of consecutive queries can be further reduced considerably.

**Keywords:** Algorithm Design and Analysis, Social Networks, Link Prediction, Network Sparsification, Query Processing, Index Structure



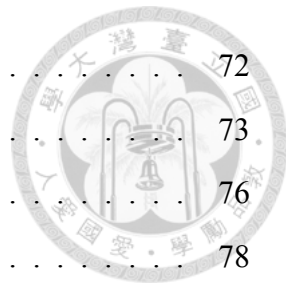
# Contents

口試委員會審定書	i
誌謝	iii
摘要	v
<b>Abstract</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Overview of the Dissertation . . . . .	1
1.1.1 Efficient Link Prediction in Large-Scale Networks . . . . .	3
1.1.2 Efficient Social-Temporal Group Query . . . . .	4
1.1.3 Efficient Consecutive Group Query Processing . . . . .	4
1.2 Organization of the Dissertation . . . . .	5
<b>2 Ensemble of Diverse Sparsifications for Link Prediction in Large-Scale Networks</b>	<b>7</b>
2.1 Introduction . . . . .	7
2.2 Related Works . . . . .	10
2.3 Framework Overview . . . . .	12
2.3.1 Proximity Measures in Link Prediction . . . . .	12
2.3.2 Supervised Framework for Link Prediction . . . . .	13
2.3.3 Data and Evaluation Metrics . . . . .	15
2.4 Diverse Ensemble of Drastic Sparsification . . . . .	15



2.4.1	Diverse Sparsification Methods . . . . .	16
2.4.2	Ensemble with Diversity . . . . .	19
2.4.3	Feature Subset Selection . . . . .	21
2.5	Strategies for Ensemble Generation . . . . .	22
2.5.1	Accuracy-Based Weight Setting . . . . .	23
2.5.2	Ensemble Size Augmentation . . . . .	26
2.6	Analysis on Accuracy and Efficiency . . . . .	27
2.6.1	Prediction Accuracy . . . . .	28
2.6.2	Computational Efficiency . . . . .	30
2.7	Summary . . . . .	31
<b>3</b>	<b>Efficient Social-Temporal Group Query with Acquaintance Constraint</b>	<b>33</b>
3.1	Introduction . . . . .	33
3.2	Related Works . . . . .	38
3.3	Social Group Query . . . . .	38
3.3.1	Problem Definition . . . . .	39
3.3.2	Algorithm Design . . . . .	41
3.4	Social-Temporal Group Query . . . . .	52
3.4.1	Problem Definition . . . . .	52
3.4.2	Algorithm Design . . . . .	53
3.5	Experimental Results . . . . .	58
3.5.1	Experiment Setup . . . . .	58
3.5.2	Performance Analysis of SGQ and STGQ . . . . .	59
3.5.3	User Study of Manual Activity Coordination . . . . .	66
3.6	Summary . . . . .	68
<b>4</b>	<b>Efficient Processing of Consecutive Group Queries for Social Activity Planning</b>	<b>69</b>
4.1	Introduction . . . . .	69
4.2	Related Works . . . . .	71

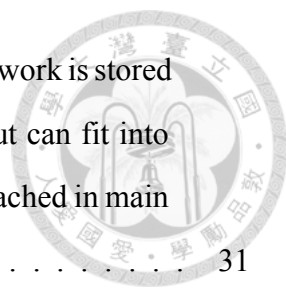
4.3	Consecutive Social Group Query . . . . .	72
4.3.1	Accumulative Search Tree and Social Boundary . . . . .	73
4.3.2	Solution Acquisition Using AST and SB . . . . .	76
4.4	Index Construction and Maintenance . . . . .	78
4.4.1	Node Indexing of AST Using SB . . . . .	78
4.4.2	Updating of AST and SB . . . . .	87
4.5	Solution Optimality and Extensions . . . . .	89
4.5.1	Solution Optimality . . . . .	89
4.5.2	Extensions in Temporal Dimension . . . . .	93
4.6	Experimental Results . . . . .	94
4.6.1	Performance Analysis of CSGQ . . . . .	95
4.7	Summary . . . . .	98
<b>5</b>	<b>Conclusion and Future Work</b>	<b>99</b>
	<b>Bibliography</b>	<b>101</b>
	<b>Appendices</b>	<b>109</b>
A	Detailed Proof . . . . .	109
A.1	Proof of Theorem 4.5.1 . . . . .	109
B	Pseudo Codes . . . . .	112





## List of Figures

2.1	Flow chart of the DEDS framework. . . . .	9
2.2	An illustrative example for the usage of the three snapshots in the DEDS framework, where $CN$ , $JC$ , and $PA$ stand for the proximity measures <i>common neighbors</i> , <i>Jaccard's coefficient</i> , and <i>preferential attachment</i> , respectively. The thick lines indicate the newly generated edges which do not exist in the preceding snapshot. . . . .	13
2.3	(a) An example network with its four different sparsified networks, (b), (c), (d), (e), obtained from degree-based sparsification, random-walk-based sparsification, short-path-based sparsification, and random sparsification, respectively. . . . .	19
2.4	Performance comparison between the ensemble without diversity and the ensemble with diversity. (condmat, sparsification ratio = 10%) . . . . .	20
2.5	Error regions associated with approximating the <i>a posteriori</i> probabilities [56]. . . . .	23
2.6	Comparison of AUC for different ensemble sizes. The dash line indicates the AUC of the classifier trained from the original network. (condmat, sparsification ratio = 15%) . . . . .	27
2.7	Performance of four different individual classifiers, their average (denoted as "Average"), and the ensemble classifier at different sparsification ratios. . . . .	29



2.8	Running time under different sparsification ratios: (a) the network is stored in the disk, (b) the original network is stored in the disk but can fit into main memory after being sparsified, and (c) the network is cached in main memory. . . . .	31
3.1	An illustrative example for STGQ. (a) The sample social network, (b) the dendrogram of candidate group enumeration and (c) schedules of candidate attendees. . . . .	35
3.2	Another illustrative example for SGQ and STGQ. (a) The sample social network, (b) the social distances of candidate attendees and (c) the schedules of candidate attendees. . . . .	50
3.3	Experimental results of SGQ. . . . .	60
3.4	Analysis on pruning ability of proposed strategies. . . . .	62
3.5	Experimental results of STGQ. . . . .	64
3.6	Experimental results with the YouTube dataset. . . . .	65
3.7	Experimental results of the user study. . . . .	66
4.1	$T_1$ and $T_2$ are two dendrograms with different $k$ , and $T_3$ is the accumulative search tree. . . . .	73
4.2	An illustrative example for CSGQ. (a) The sample social network, (b) the initial accumulative search tree and (c) the accumulative search tree after the second query. . . . .	77
4.3	Experimental results of CSGQ. . . . .	96



# List of Tables

2.1	Statistics of network characteristics for different sparsified networks. . . . .	19
3.1	The percentage of prunings located near the root of the dendrogram (i.e., with $ V_S  \leq \lfloor \frac{p}{2} \rfloor$ ). . . . .	63
4.1	The $(s,k)$ -SBs constructed after the first SGQ. . . . .	76
4.2	(a) The social distances from different vertices to $v_7$ under various $s$ , and (b) variations of node $P5$ . . . . .	83
4.3	The $(s,k)$ -SBs after updated according to the second SGQ. . . . .	88





# Chapter 1

## Introduction

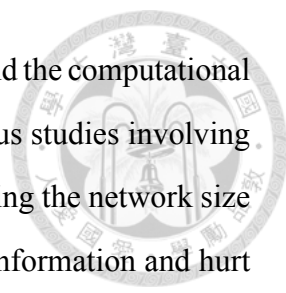
### 1.1 Motivation and Overview of the Dissertation

As the development and popularization of social networking websites, many recommendation systems tend to leverage the information in social networks to provide helpful suggestions for users, and a great deal of research studies on social network analysis are thereby motivated. For example, link prediction can recommend movies or restaurants to users based on their friends' opinions, and it can also suggest people that users may be friend with (e.g., the “people you may know” feature in Facebook and LikedIn). Recently, the sizes of social networks have been increasing rapidly, and this growth results in a significant increase in the computational cost of the sophisticated recommendations. For example, the number of monthly active users on Facebook reaches 1.79 billion in 2016.<sup>1</sup> The huge size and complexity of social networks create a considerable burden for recommendation systems while processing the information from social networks to provide suggestions. Therefore, in this dissertation, we study three important recommendation problems in social networks and aim to improve their efficiency. The challenges of these recommendation problems are introduced below.

First, we focus on the relationship between two users and study the link prediction problem in large-scale networks. During the link prediction, numerous feature values

---

<sup>1</sup>The statistics provided by Facebook. <http://newsroom.fb.com/company-info/>.



need to be calculated and then combined to make recommendations, and the computational cost grows quickly as the network size becomes larger. Some previous studies involving network processing attempt to lower the computational cost by reducing the network size via sparsification. However, sparsification might remove important information and hurt the prediction accuracy. Therefore, the primary challenge is to reduce the network size considerably while maintaining high prediction accuracy.

Second, we extend the scope from the relationship between two users to the relationship among a group of users, and study the social-temporal group query problem with its applications in activity planning. While the first problem we study (i.e., link prediction) focuses on predicting the existence of a link between particular two users, the second problem considers existing links among all users to recommend a mutually acquainted group of attendees for an activity. This is an NP-hard problem, and the computational cost also grows rapidly as the network size increases. In addition to finding a group of attendees familiar with each other, selecting an activity period available to all attendees is also essential for activity planning. Therefore, we need to further consider the available time of users, which makes the problem even harder due to the complexity of social connectivity and the diversity of user schedules.

Third, we study the consecutive group query problem to support a sequence of recommendations. When planning an activity, it is difficult for a user to specify all the conditions right at once to find the perfect group of attendees and time. Fortunately, with the aforementioned social-temporal group query, it is easy for the user to tune the parameters to obtain alternative recommendations. Allowing tuning parameters to try consecutive queries easily is a great advantage of the planning service over the current practice of manual planning. However, answering each of the consecutive queries individually will lead to repeated exploration of similar solution space, since these queries are issued by the same user with slightly adjusted parameters. Therefore, new challenges arise in the design of an effective index structure for maintenance and examination of the intermediate results to facilitate efficient processing of consecutive queries.

In the following, we provide overviews for the aforementioned three problems studied

in this dissertation and the proposed solutions.



### 1.1.1 Efficient Link Prediction in Large-Scale Networks

Since managing massive networks is complex and time-consuming, various studies have focused on network sparsification, simplification, and sampling [16] [47] [25] [33] [45] [54]. Many of these algorithms are designed to preserve certain properties of interest while reducing the size of networks, so that the sparsified or simplified networks may remain informative for future targeted applications. For example, the simplification algorithm in [16] is designed as a preprocessing step prior to network visualization, while the sparsification algorithm in [47] is designed to sparsify the network before clustering. These existing algorithms are effective in their target applications. However, to the best of our knowledge, none of these works has been specifically designed with classifier ensembling to facilitate link prediction. Such algorithms may remove the part of the network that is informative for link prediction, and hence lead to a substantial decrease in prediction accuracy.

To address this issue, we propose a framework called *Diverse Ensemble of Drastic Sparsification (DEDS)*, which constructs ensemble classifiers with good accuracy while keeping the prediction time short. DEDS includes various sparsification methods that are designed to preserve different measures of a network. Therefore, DEDS can generate sparsified networks with significant structural differences and increase the diversity of the ensemble classifier, which is key to improving prediction performance. According to the experimental results, when a network is drastically sparsified, DEDS effectively relieves the drop in prediction accuracy and raises the AUC value. With a larger sparsification ratio, DEDS can even outperform the classifier trained from the original network. As for the efficiency, the prediction cost is substantially reduced after the network is sparsified. If the original network is disk-resident but can fit into main memory after being sparsified, the improvement is even more significant.



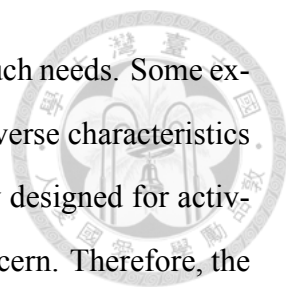
### 1.1.2 Efficient Social-Temporal Group Query

For social activity planning, three essential criteria are important: (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 dissertation, we propose the *Social-Temporal Group Query (STGQ)* to find suitable time and attendees with minimum total social distance. By minimizing the total social distance among the attendees, we are actually forming a cohesive subgroup in the social network. In the field of social network analysis, research on finding various kinds of subgroups, such as clique, k-plex and k-truss has been conducted (e.g., [6, 18, 43, 55, 59]). There are also related works on group formation (e.g., [3, 44, 57]), team formation (e.g., [2, 24, 41]), and group query (e.g., [27, 28, 60]). While these works focus on different scenarios and aims, none of them simultaneously encompass the social and temporal objectives to facilitate automatic activity planning. Therefore, the STGQ problem is not addressed previously.

In our study of STGQ, we first prove that the problem is NP-hard and inapproximable within any ratio. Next, we design two algorithms, *SGSelect* and *STGSelect*, which include various effective pruning techniques to substantially reduce running time. Experimental results indicate that *SGSelect* and *STGSelect* are significantly more efficient and scalable than the baseline approaches. Our research results can be adopted in social networking websites and web collaboration tools as a value-added service. We also conduct a user study to compare the proposed approach with manual activity coordination. The results show that our approach obtains higher quality solutions with less coordination effort, thereby increasing users' willingness to organize activities.

### 1.1.3 Efficient Consecutive Group Query Processing

According to the feedbacks from the user study we conduct, it is difficult for an activity initiator to specify all the conditions right at once to find the perfect group of attendees and time, and hence the initiator tends to tune the parameters to find alternative solutions. As users may iteratively adjust query parameters to fine tune the results, we further study



the problem of *Consecutive Social Group Query (CSGQ)* to support such needs. Some existing studies (e.g., [14, 34, 63, 64]) return multiple subgraphs with diverse characteristics in one single query. However, since these studies are not specifically designed for activity planning, social connectivity and tightness are not their major concern. Therefore, the returned subgroups are not guaranteed to achieve social cohesiveness. Moreover, without feedback and guidance from user-specified parameters, most returned subgraphs in the diversified query are likely to be redundant (i.e., distant from the desired results of users). On the other hand, session query and reinforcement learning in retrieval (e.g., [20, 26, 50]) that allow users to tailor the query have attracted increasing attentions. However, these studies are designed for document retrieval and hence cannot handle the social network graph and user schedules. Therefore, these aforementioned research works are not applicable for automatic activity planning, and the CSGQ problem is not addressed previously.

Anticipating that the users would not adjust the parameters drastically, we envisage that exploiting the intermediate solutions of previous queries may improve processing of succeeding queries. In our study of CSGQ, we design two new data structures to facilitate the above idea and efficiently support a sequence of group queries with varying parameters. We first design a new tree 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 a new index structure, called *Social Boundary*, which effectively indexes the intermediate solutions required for processing each CSGQ with specified parameters. According to the experimental results, with the caching mechanisms, processing time of consecutive queries can be further reduced considerably.

## 1.2 Organization of the Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we propose the DEDS framework consisting of four different sparsification methods and show that, while using only a small portion of the edges causes considerable performance deterioration, our ensemble classifier with high diversity can counter the drop in prediction accuracy. In

Chapter 3, we formulate STGQ for automatic activity planning, and propose Algorithm SGSelect and Algorithm STGSelect with various strategies to find the optimal solution efficiently. In Chapter 4, we further introduce CSGQ, and then design new data structures to avoid redundant exploration of solution space and speed up the processing of consecutive queries. Finally, Chapter 5 concludes this dissertation and presents the future directions.



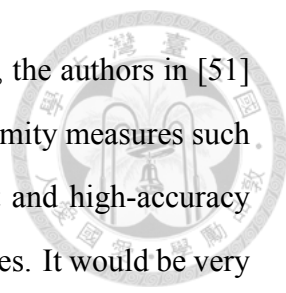
## Chapter 2

# Ensemble of Diverse Sparsifications for Link Prediction in Large-Scale Networks

### 2.1 Introduction

Link prediction is to forecast the existence of a link between two vertices, and it is an important research topic in network analysis, since many major problems involving networks can benefit from the process. Recently, the sizes of networks have been increasing rapidly, and this growth results in a significant increase in the computational cost of link prediction. Moreover, these networks may become too large to be stored in main memory. Consequently, processing these networks requires frequent disk access, which may lead to considerable deterioration in performance. As a result, prediction tasks can take days to complete, meaning that dynamic friend suggestions or product recommendations cannot be made to users or customers in a timely manner, and the recommendations may therefore become less useful as time passes.

In link prediction, there are several measurements, known as *proximity measures*, used to indicate how likely it is for a non-neighboring vertex pair to be connected via an edge in the near future. A possible solution for speeding up link prediction is to design algo-



rithms to approximate each of the proximity measures. For example, the authors in [51] and [49] proposed methods to achieve a close approximation of proximity measures such as *Katz* [22] and *rooted PageRank* [29]. However, building robust and high-accuracy classifiers for link prediction often requires various proximity measures. It would be very complicated if we were to design different algorithms to approximate each of these proximity measures. Therefore, a general and flexible solution for lowering computational costs is required.

Inspired by previous research that simplifies large networks to decrease computational costs, we found that reducing the size of networks provides a more general solution.<sup>1</sup> Once a network has been sparsified, most proximity measures can benefit from the size reduction of the network and can be calculated faster. If the network is too large to be stored in main memory, decreasing its size also helps to lower the number of disk accesses. Furthermore, when the sparsification ratio is sufficiently small, the sparsified network may be able to fit into main memory, which means the burden of disk access is relieved. However, with such drastic sparsification, many edges in the network are removed, and the information that can be used in link prediction becomes rather limited. In turn, the prediction accuracy would drop significantly under such severe conditions. Therefore, the primary challenge is to reduce the network size considerably while maintaining high prediction accuracy.

In this chapter, we address this issue by proposing a sparsification framework for link prediction called *Diverse Ensemble of Drastic Sparsification (DEDS)*, which consists of sparsifying, training, and ensembling, as shown in Figure 2.1. Specifically, we design four different methods to sparsify the original network, train individual classifiers from the sparsified networks, and ensemble these classifiers appropriately to improve prediction performance. The rightmost sparsified network in Figure 2.1 is obtained from the most straightforward random sparsification. In addition, DEDS incorporates three more sophisticated sparsification methods, which are based on heuristics for preserving the edges required by different proximity measures. DEDS is able to generate sparsified networks

---

<sup>1</sup>Certain previous studies have proposed methods that remove vertices and edges to simplify the network (e.g., [25] and [45]), while other methods only remove edges (e.g., [33] and [47]). In our study, we do not remove vertices, since any vertex may be the target that we want to generate a prediction for.



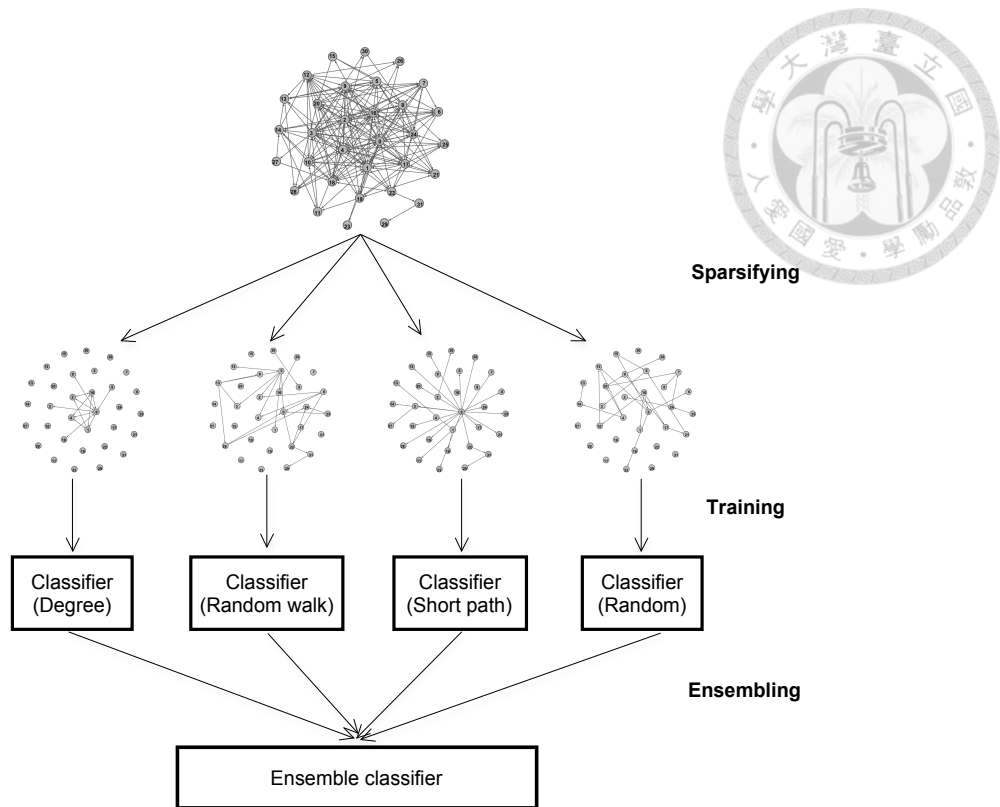


Figure 2.1: Flow chart of the DEDS framework.

with significant structural differences, and this increases the diversity of the correspondingly trained classifiers, which is key to creating an effective ensemble classifier. As shown in the experimental results, the proposed DEDS framework can effectively relieve the drop in prediction accuracy, while considerably reducing running time.

The main contributions of this chapter are summarized as follows.

- We propose a novel network sparsification framework called DEDS to slim down large networks while preserving important proximity measures that are used in link prediction. Specifically, we design four different sparsification methods by categorizing the proximity measures and then devising heuristics to preserve the edges required by these measures. The proposed DEDS framework can generate sparsified networks with significant structural differences and increase the diversity of the ensemble classifier. We also prove that adopting accuracy-based weighting enables DEDS to further reduce the prediction error. Experimental results show that when the network is drastically sparsified, DEDS can effectively relieve the drop in prediction accuracy and considerably raise the AUC value. With a larger sparsification

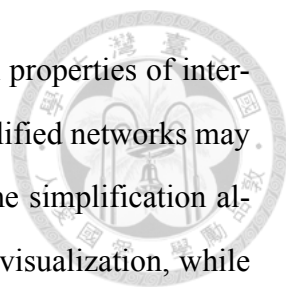
ratio, DEDS can even outperform the classifier trained from the original network.

- Our proposed DEDS framework is able to significantly reduce the running time of link prediction tasks. According to the experimental results, the prediction cost is substantially reduced after the network is sparsified. Moreover, if the network is too large to be stored in main memory, DEDS helps to lower the number of disk accesses by reducing the network size. When the sparsification ratio is sufficiently small, DEDS provides further efficiency by relieving the burden of disk access, since the sparsified network can fit into main memory.
- In the proposed DEDS framework, all the individual classifiers remain unentangled before the final decision is generated, meaning that each individual classifier can be trained and run independently. This enables DEDS to fully utilize all the CPUs or cores to simultaneously train and run the maximum number of individual classifiers. As a result, DEDS can maximize the ensemble size based on the user's computational ability and considerably increase prediction accuracy.

The rest of this chapter is organized as follows. In Section 2.2, we introduce related works. Section 2.3 provides preliminaries for a supervised framework of link prediction, and describes the datasets and evaluation metrics used throughout this study. In Section 2.4, we propose four different sparsification methods and show that, while using only a small portion of the edges causes considerable performance deterioration, our ensemble classifier with high diversity can counter the drop in prediction accuracy. In Section 2.5, we analyze two strategies which further raise the performance of the ensemble classifier. More detailed experimental results, such as efficiency analysis, are provided in Section 2.6. We summarize this chapter in Section 2.7.

## 2.2 Related Works

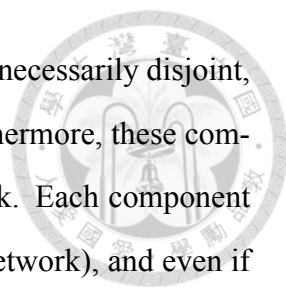
Since managing massive networks is complex and time-consuming, various studies have focused on network sparsification, simplification, and sampling [16] [47] [25] [33]



[45] [54]. Many of these algorithms are designed to preserve certain properties of interest while reducing the size of networks, so that the sparsified or simplified networks may remain informative for future targeted applications. For example, the simplification algorithm in [16] is designed as a preprocessing step prior to network visualization, while the sparsification algorithm in [47] is designed to sparsify the network before clustering. These existing algorithms are effective in their target applications. However, to the best of our knowledge, none of these works has been specifically designed with classifier ensembling to facilitate link prediction. Such algorithms may remove the part of the network that is informative for link prediction, and hence lead to a substantial decrease in prediction accuracy. Moreover, the existing network sparsification algorithms are not designed to deal with drastic sparsification, and most of them have used a sparsification ratio in the range of tens of percent. In our study, we make use of even smaller sparsification ratios (e.g., 5%) and explain how prediction accuracy can be maintained under such severe conditions.

Certain previous studies have succeeded in preserving one of the proximity measures used in the link prediction task. For example, the method in [45] can recover the shortest-path distance between a vertex pair from the simplified graph, while that in [54] can preserve the quality of the best path between a vertex pair. However, various proximity measures are required to achieve high accuracy in link prediction; thus, merely focusing on a single measure is not sufficient. In this chapter, we first discuss the properties of the different proximity measures used in link prediction, and then we propose a novel sparsification framework that aims to preserve the discrimination ability of these measures to provide good prediction accuracy.

Another area of related work is graph partitioning, which includes a focus on handling large-scale graphs. Many useful graph partitioning tools already exist (e.g., Chaco [15] and METIS [21]), and certain recent studies continue to focus on graph partitioning [62] [53]. A common goal of graph partitioning is to divide a large graph into smaller disjoint components of approximately the same size, which together can cover the entire original graph. In our work, we also create multiple small and even-sized components from



the original large-scale network. However, these components are not necessarily disjoint, and some informative edges may exist in multiple components. Furthermore, these components together are not required to cover the entire original network. Each component may include only a small portion of edges (e.g., 1% of the original network), and even if tens of components are created, the edges used are still less than the edges in the original network.

## 2.3 Framework Overview

Before introducing the details of the DEDS framework, we provide in Section 2.3.1 preliminaries on the proximity measures that are commonly used in link prediction tasks. We also introduce in Section 2.3.2 the method for leveraging these proximity measures under a supervised link prediction framework. In Section 2.3.3, we describe the datasets and the evaluation metrics used throughout this chapter.

### 2.3.1 Proximity Measures in Link Prediction

Most existing link prediction methods involve calculating proximity measures for a non-neighboring vertex pair,  $v_i$  and  $v_j$ , where the higher the measures are, the more likely  $v_i$  and  $v_j$  are to be connected via an edge in the near future. According to [29], most basic link prediction methods generate proximity measures based on the neighborhood information of  $v_i$  and  $v_j$ , or on the path information between  $v_i$  and  $v_j$ . The methods that rely on neighborhood information include *common neighbors*, *Jaccard's coefficient* [46], *Adamic/Adar* [1], and *preferential attachment* [5]. The common neighbors method calculates the number of neighbors that  $v_i$  and  $v_j$  have in common, while the Jaccard's coefficient method modifies the common neighbors method by normalizing it with the total number of neighbors that  $v_i$  and  $v_j$  have. The Adamic/Adar method modifies the common neighbors method by giving more weight to the neighbor that is rarer (i.e., the neighbor that is connected to fewer other vertices). In the preferential attachment method, the proximity measure is determined by multiplying the number of neighbors of  $v_i$  and  $v_j$ .

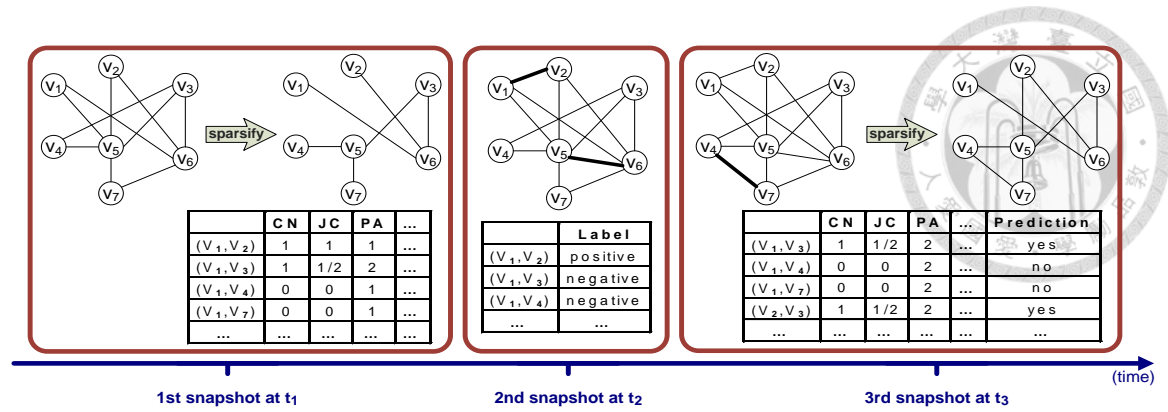


Figure 2.2: An illustrative example for the usage of the three snapshots in the DEDS framework, where  $CN$ ,  $JC$ , and  $PA$  stand for the proximity measures *common neighbors*, *Jaccard's coefficient*, and *preferential attachment*, respectively. The thick lines indicate the newly generated edges which do not exist in the preceding snapshot.

The methods that rely on path information include *Katz* [22], *hitting time* [29], variants of *PageRank* [7] (e.g., *rooted PageRank* [29]), *SimRank* [19], and *PropFlow* [31]. A description and comparison of the methods listed above (with the exception of the recently proposed *PropFlow*) can be found in [29].

### 2.3.2 Supervised Framework for Link Prediction

In the proposed DEDS framework, we adopt the common proximity measures mentioned in the previous subsection within a supervised framework. As shown in [31], using scores generated from the same proximity measure, a supervised method can outperform an unsupervised method. A supervised method is also more capable of handling the dynamics and capturing the interdependency of topological properties in the network. Therefore, recent link prediction works have begun to use a supervised framework.

A supervised framework like DEDS works as illustrated in Figure 2.2. The first snapshot of the original network is taken to generate feature values (i.e., the scores generated from proximity measures). Before computing the proximity measures of vertex pairs in the first network snapshot, we sparsify the snapshot using the proposed sparsification methods, which will be described later in Section 2.4.1. The computational cost of calculating proximity measures can be reduced considerably, since the network size is significantly lowered. Note that before sparsifying the network snapshot, the non-neighboring vertex

pairs are marked in main memory or the disk.<sup>2</sup> This prevents unnecessary prediction of the vertex pairs that are originally connected but have their connecting edges removed during sparsification. For example, even though the edge between the vertex pair  $(v_1, v_5)$  is removed during sparsification, we can still avoid making a redundant prediction on this vertex pair.

After a given period, a second snapshot of the network is taken.<sup>3</sup> Since the network is dynamic, this second snapshot can be used to verify whether an originally non-neighboring pair  $v_i$  and  $v_j$  in the first snapshot is now connected or not, and class labels are thereby generated. Taking the example illustrated in Figure 2.2, the vertex pair  $(v_1, v_2)$  is newly connected in the second snapshot, and hence this pair is a positive instance. While the first snapshot is sparsified in order to generate proximity measures efficiently, the second snapshot is fully preserved. This ensures that the class labels generated from the second snapshot are all correct.

Besides the first and the second network snapshots, there is a third snapshot used to capture the most recent network. We use this last network snapshot to calculate the latest proximity measures and make predictions for users. Since the goal of the drastic sparsification is to generate the prediction much faster, the third snapshot is sparsified before calculating the proximity measures. Some may think that the training process using the first and second network snapshots can be conducted offline without rushing, so we do not need to sparsify the first network snapshot. However, the sparsification of the first snapshot is actually necessary and even vital to prediction accuracy. If the first snapshot is not sparsified, most proximity measures, such as the common neighbors and the Katz, tend to have much higher values. As a consequence, the classifier trained with these over-estimated values cannot make accurate predictions in the third network snapshot, which is sparsified and tends to have lower values.

---

<sup>2</sup>With the help of data structures such as B-tree, we can decide quickly whether a vertex pair has a link in the original network or not, even if this information is stored in the disk.

<sup>3</sup>The timing for taking these snapshots of the network (i.e.,  $t_1$  and  $t_2$ ) is discussed in [31].



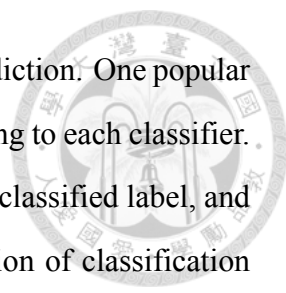
### 2.3.3 Data and Evaluation Metrics

To evaluate the improvement of prediction accuracy introduced by the DEDS framework, we use two real datasets provided in the LPmade package [30]: *condmat* (110.5K edges) and *disease* (15.6K edges). Besides the real datasets, we also carry out the evaluation of efficiency on a larger synthetic dataset *rmat* (10M edges), which is produced using the *R-MAT* [8] graph generating algorithm. The R-MAT algorithm is able to quickly generate large networks that match patterns in real-world networks. The parameters  $a$ ,  $b$ ,  $c$ , and  $d$  used in network generation are set to be 0.6, 0.15, 0.15, and 0.1, which follow the settings in the previous study [32].

In terms of evaluation metrics, the receiver operating characteristic (ROC) curve is an effective method for illustrating the performance of a binary classifier system, since it shows the performance of the classifier under the entire operating range. The ROC curve displays the true positive rate (TPR) versus the false positive rate (FPR), and each point on this curve is produced by varying the decision threshold of the proximity measures. Since it would be too space-consuming to generate ROC curves for all of the classifiers under various sparsification ratios, the following evaluation is based on the area under the ROC curve (AUC). AUC is a related scalar measure of ROC, and it can be seen as a summary of performance. Besides using AUC to evaluate the performance over all decision thresholds, we also adopt the common mean square error (MSE) to measure the performance at a specific decision threshold that is selected during the training phase.

## 2.4 Diverse Ensemble of Drastic Sparsification

When only a small portion of edges are preserved during sparsification, the information that can be used in link prediction becomes extremely limited. Therefore, the challenge is to maintain high prediction accuracy while drastically reducing the network size. The proposed DEDS framework solves this problem by using the wisdom of crowds; that is, ensemble. The concept of ensemble is to build different experts (i.e., individual classifiers) and then let them vote. Just as considering various opinions may help people make better



decisions, using different classifiers allows for more accurate link prediction. One popular and simple ensemble method is bagging, which assigns equal weighting to each classifier. Bagging can be used to average the numerical outputs or vote for the classified label, and it provides the DEDES framework with more flexibility in the selection of classification algorithms. In this study, we present the results with the classic *C4.5* [40], and the results with *naive Bayes* show similar trends. However, if a particular classification algorithm is found to be more suited to handling a certain network, users can easily replace the current algorithm used in DEDES without modifying the framework structure.

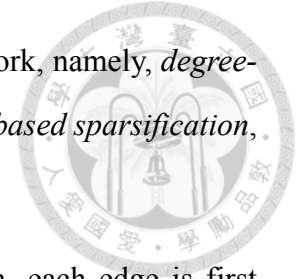
To obtain an effective ensemble classifier, previous studies [56] [17] [23] suggest that the diversity among individual classifiers plays an important role. Without diversity, the individual classifiers tend to produce similar judgments, and combining these similar judgments will not improve the final decision. Therefore, in order to increase diversity and preserve various important properties, DEDES incorporates four sparsification methods to generate a variety of sparsified networks. We introduce these sparsification methods in Section 2.4.1, and in Section 2.4.2, we analyze how the ensemble with diversity can result in a better performance. In Section 2.4.3, we further accelerate the link prediction process by feature selection.

### **2.4.1 Diverse Sparsification Methods**

In the DEDES framework, the proximity measures described in Section 2.3 are adopted as features for building the classifier for link prediction. Since these proximity measures have different properties, a single type of sparsified network is unable to sufficiently preserve them all. Therefore, we generate various types of sparsified networks, and in each network, certain proximity measures can benefit from the edges preserved. As a result, these proximity measures can be more discriminative. Furthermore, as previously mentioned, diversity is an important factor in obtaining an effective ensemble. By generating different sparsified networks with various sparsification methods, we can increase the variety of classifiers trained from these sparsified networks. In the following paragraphs, we



introduce the four sparsification methods used in the DEDS framework, namely, *degree-based sparsification*, *random-walk-based sparsification*, *short-path-based sparsification*, and *random sparsification*.



- **Degree-based sparsification.** In degree-based sparsification, each edge is first given a score in proportion to the summation of the degrees at its two ends. We then repeatedly select the edge with the highest score, until the percentage of selected edges meets the sparsification ratio. As mentioned, the proximity measures adopted in the DEDS framework can be roughly divided into two categories: one based on neighborhood information, and the other based on path information. Most proximity measures based on neighborhood information involve counting the number of common neighbors. To further illustrate this, let us assume there is a connected vertex pair of  $v_x$  and  $v_y$ . During sparsification, if the degree of either  $v_x$  or  $v_y$  is large, removing the edge connecting the pair may affect many common neighbor counts. Let the degree of  $v_x$  to be  $d_x$ , which means  $v_x$  has  $d_x - 1$  neighbors other than  $v_y$ . When we remove the edge connecting  $v_x$  and  $v_y$ , the number of common neighbors between  $v_y$  and each of the  $d_x - 1$  neighbors of  $v_x$  will decrease by one, since  $v_x$  is no longer a neighbor of  $v_y$  after sparsification. For the  $d_y - 1$  neighbors of  $v_y$ , the common neighbor  $v_x$  also disappears between  $v_x$  and each of these  $d_y - 1$  vertices. In total, removing the edge connecting  $v_x$  and  $v_y$  may cause as many as  $d_x + d_y - 2$  non-neighborhood vertex pairs to lose their common neighbors.<sup>4</sup> Therefore, we choose to remove the edge with the smaller summation of the degrees at its two ends, in order to reduce the loss of common neighbors.

- **Random-walk-based sparsification.** Certain proximity measures based on path information (e.g., hitting time, rooted PageRank, and PropFlow) involve random walks between non-neighborhood vertices. Therefore, in this sparsification method, we aim to preserve the edges that are most frequently used by the random walk.

We first conduct random walk rehearsals on randomly selected vertex pairs in the

---

<sup>4</sup>There are  $d_x + d_y - 2$  vertex pairs that lose common neighbors, but only the non-neighborhood vertex pairs will affect prediction accuracy. Therefore,  $d_x + d_y - 2$  is actually a worst-case analysis.

original network, and then we calculate the score of each edge in proportion to the total visited count during the random walk rehearsals. Thereafter, we repeatedly select the edge with the highest score, until the percentage of selected edges meets the sparsification ratio.

- **Short-path-based sparsification.** In addition to random walk, certain proximity measures based on path information (e.g., Katz) involve short paths between non-neighboring vertices. Therefore, in this sparsification method, we aim to preserve the edges that appear frequently in certain short paths, as these edges are likely to be shortcuts or important bridges. We first compute the shortest paths that do not exceed a length threshold  $L$ , and the score of each edge is calculated in proportion to the frequency that this edge appears in the paths. We then repeatedly select the edge with the highest score, until the percentage of selected edges meets the sparsification ratio. In most previous studies of link prediction, the path-oriented proximity measures do not involve long paths since they are computationally expensive, and using the maximum length of the paths used in proximity measures (e.g., 5 in [31]) as  $L$  will suffice. When the original network is too large to be stored in main memory, the *sketch-based index structure* proposed in [13] can help to compute the paths much faster, while keeping the estimation error below 1% on average.

- **Random sparsification.** The random sparsification method is the most straightforward among the four methods, whereby edges are randomly selected from the original network until the percentage of selected edges meets the sparsification ratio. This sparsification prevents the DEDS framework from being over-fitted for any specific type of proximity measure, and also allows the DEDS framework to be more generalized to accommodate potential new proximity measures in the future.

Since the original network is comparatively large, the sparsification process that deals with the original network may take longer. However, the sparsification can be done offline, and then predictions can be generated online in a timely fashion using the smaller sparsified network.

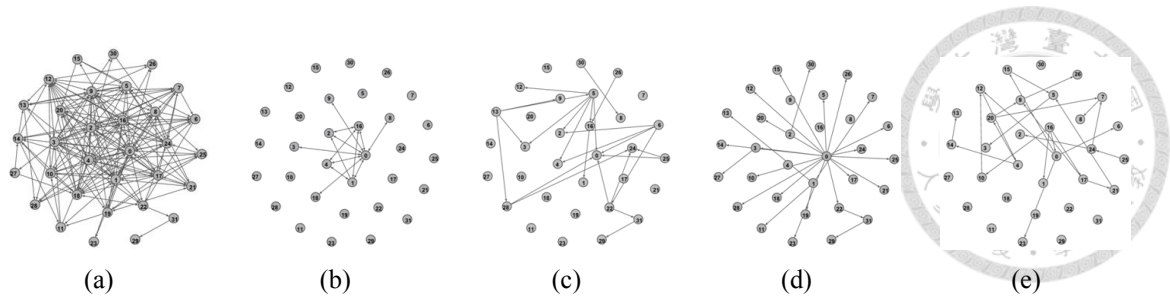


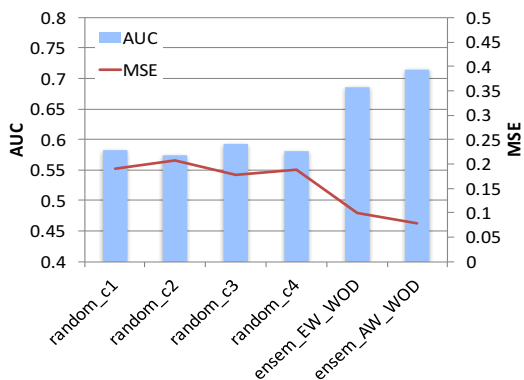
Figure 2.3: (a) An example network with its four different sparsified networks, (b), (c), (d), (e), obtained from degree-based sparsification, random-walk-based sparsification, short-path-based sparsification, and random sparsification, respectively.

Table 2.1: Statistics of network characteristics for different sparsified networks.

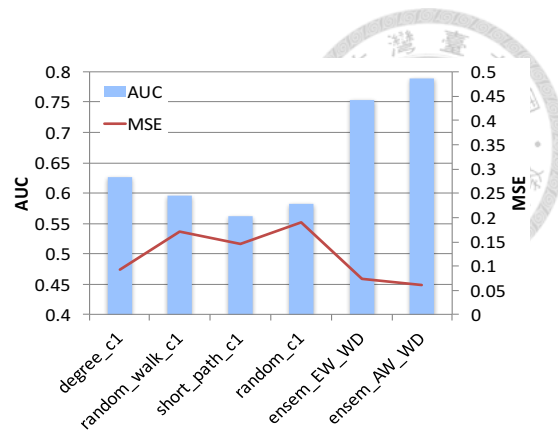
Characteristics	Mean		SD		RSD	
	random_spar	diverse_spar	random_spar	diverse_spar	random_spar	diverse_spar
Assortativity Coef.	-0.244	-0.337	0.018	0.292	7%	86%
Average Clustering Coef.	0.033	0.097	0.006	0.067	17%	69%
Median Degree	2	0.5	0	1	0%	200%
Max Degree	23.25	49.75	0.96	23.82	4%	48%
Number of SCCs	116	258	7.26	90.41	6%	35%
Largest SCC	283.25	129	7.68	104.09	3%	81%
Largest SCC Diameter	7.5	5.5	1.29	2.38	17%	43%

## 2.4.2 Ensemble with Diversity

Figure 2.3 displays an example network and its four sparsified networks obtained by the sparsification methods introduced in Section 2.4.1. It can be seen that these sparsified networks have significant structural differences. These sparsified networks also exhibit different quantitative characteristics. Table 2.1 provides various network statistics to contextualize the diversity of these sparsified networks. Here we compare two groups of sparsified networks, namely, `random_spar` (which includes four randomly sparsified networks) and `diverse_spar` (which includes four sparsified networks obtained from different sparsification methods). For each sparsified network, we calculate various characteristics as described below. To measure the tendency of finding connected vertex pairs that are each highly connected, we use the assortativity coefficient. In order to measure how likely vertices in the network are to be connected in dense groups, we use the average clustering coefficient. To more fully grasp the broad topological structure of the network, we also calculate the size and diameter of the strongly-connected components (SCCs). For simplicity and easy comparison, we do not list the values for each of the sparsified networks separately. Instead, we provide summarized statistics for the four sparsified networks in



(a) Four individual classifiers generated from the same type of sparsified networks, and two ensemble classifiers without diversity (shown from left to right).



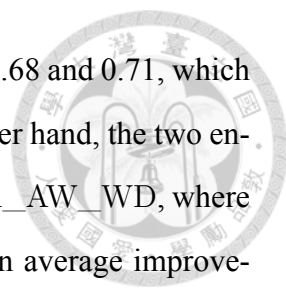
(b) Four individual classifiers generated from different types of sparsified networks, and two ensemble classifiers with diversity (shown from left to right).

Figure 2.4: Performance comparison between the ensemble without diversity and the ensemble with diversity. (condmat, sparsification ratio = 10%)

each group, which include the mean, the standard deviation (SD), and the relative standard deviation (RSD). The RSD is widely used to measure the degree of dispersion, and a larger RSD means the data has more variability. As the results show, the RSD of *diverse\_spar* is always significantly higher than the RSD of *random\_spar*, which indicates that the network characteristics of the sparsified networks obtained from different sparsification methods are much more diverse.

In Figure 2.4, experimental results are provided to compare the ensemble without diversity and the ensemble with diversity. Here we present the results with condmat, and the other real dataset disease exhibits similar trends. In Figures 2.4(a) and 2.4(b), there are seven individual classifiers (i.e., random\_c1-4, degree\_c1, random\_walk\_c1, and short\_path\_c1, where the prefix of each classifier indicates its corresponding type of sparsified network), and four ensemble classifiers composed of these individual classifiers. Note that the AUC of the individual classifiers is relatively low, with most of them under 0.6. In contrast, if we combine these weak individual classifiers, even the AUC of the worst ensemble classifier can exceed 0.68, which is an improvement of 17% compared with the average of its four individual classifiers.

We can further note that there exists a large difference among the ensemble classifiers. The two ensemble classifiers in Figure 2.4(a) (i.e., ensem\_EW\_WOD and en-



sem\_AW\_WOD, where WOD means without diversity) can reach 0.68 and 0.71, which is an average improvement of 17% and 22%, respectively. On the other hand, the two ensemble classifiers in Figure 2.4(b) (i.e., ensem\_EW\_WD and ensem\_AW\_WD, where WD means with diversity) can reach 0.75 and 0.79, which shows an average improvement of 27% and 34%, respectively.<sup>5</sup> The ensemble classifiers in Figure 2.4(b) have no larger ensemble size than the ensemble classifiers in Figure 2.4(a), and the key difference that makes the former outperform the later is the diversity. Specifically, the four individual classifiers composing the ensemble classifiers in Figure 2.4(b) are trained from four different types of sparsified networks with significant structural differences, as shown in Figure 2.3 and Table 2.1. In addition to AUC, the ensemble classifiers with higher diversity also have better MSE values. As shown by the experimental results, the average MSE of the ensemble classifiers in Figure 2.4(b) is reduced by 25% as compared with the average MSE of the ensemble classifiers in Figure 2.4(a). The above analysis indicates that, although the total number of edges preserved in the sparsified networks are limited, ensembling with various types of sparsified networks can increase the diversity to help relieve the drop in accuracy and maintain a solid performance.

### 2.4.3 Feature Subset Selection

As shown in Section 2.4.2, different types of sparsified networks are diverse and have significant structural differences. Therefore, the features (i.e., the proximity measures) are not likely to be equally effective in the correspondingly trained classifiers. For the applications that need to provide online predictions efficiently, using feature selection to form a proper feature subset for each classifier can further accelerate the prediction and reduce the delay. Among the existing feature selection algorithms, the category of wrapper methods usually outperforms the category of filters methods, since the former evaluates feature subsets based on the adopted prediction model. However, most wrapper methods need to test numerous combinations of features and hence become computationally in-

---

<sup>5</sup>In our study, we also found that weight setting will affect performance, and the difference between equal weighing (denoted with EW) and accuracy-based weighting (denoted with AW) will be analyzed later in Section 2.5.1.

tensive, especially when we have multiple classifiers to train. Therefore, we consider a simple alternative, which evaluates the performance of each feature in different sparsified networks separately.

Specifically, in the preprocessing stage, each feature will have four scores determined by its discrimination ability (e.g., AUC) in the four different sparsified networks. Later in the training process, if any feature has one score below a specified threshold, this feature will not be adopted to make prediction in the corresponding type of sparsified network. In this way, only the features have enough discrimination ability for this type of sparsified network are kept to build the classifier. An intuitive threshold is the average of all the scores, which picks out features with relatively stronger discrimination ability. If the user wants to further accelerate the online prediction, the threshold can be raised to reduce the number of features.

According to the experimental results, embedding feature selection as a preprocessing stage saves 21% on running time on average, ranging from 15% to 37%. Meanwhile, the AUC of the ensemble classifier only slightly drops by 0.007, which is less than 1%. The results indicate that the feature selection effectively preserves the features with high discrimination ability in each sparsified network, and therefore is able to reduce considerable running time without sacrificing much on the prediction accuracy. Moreover, when new features are proposed in the future, it is difficult for users without expert knowledge to decide which ones are worth adopting. The feature selection here can help automatically determine whether the new features perform well enough in a certain sparsified network, and it can also eliminate the outdated features to save computational cost. Therefore, incorporating the feature selection also enhances the flexibility to adopt new features in DEES.

## 2.5 Strategies for Ensemble Generation

As shown in Section 2.4, the ensemble classifier with high diversity can greatly outperform every individual classifier. In this section, we analyze two strategies that further

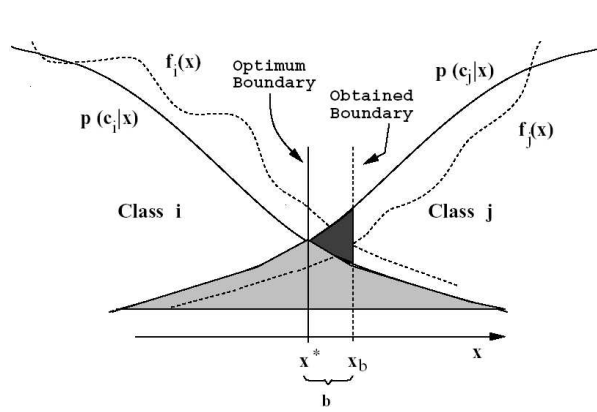


Figure 2.5: Error regions associated with approximating the *a posteriori* probabilities [56].

raise its performance: using the accuracy-based weighting in Section 2.5.1 and augmenting the ensemble size in Section 2.5.2.

### 2.5.1 Accuracy-Based Weight Setting

While the bagging method assigns equal weighting to each classifier, we found that if the weights of classifiers are carefully assigned, the performance of DEDS can be further improved. With the following proof, we show that the weights of classifiers should be assigned in proportion to their prediction accuracy.

**Theorem 2.5.1.** *The DEDS framework adopting accuracy-based weighting introduces a smaller prediction error than the framework adopting equal weighting.*

*Proof.* Since the output of a reasonably well-trained classifier is expected to approximate the corresponding *a posteriori* class distribution, the obtained decision boundary is expected to be close to the Bayesian decision boundary. In a two-class classification problem such as link prediction, the Bayesian optimum decision assigns an instance  $x$  to the class  $i$  if  $p(c_i|x) > p(c_j|x)$ , where  $p(c|x)$  is the *a posteriori* probability distribution of class  $c$  given the input  $x$ . In other words, the Bayes optimum boundary is at the point  $x^*$  such that  $p(c_i|x^*) = p(c_j|x^*)$ . However, the trained classifier is not perfect, and it outputs  $f_c(y) = p(c|x) + \eta_c(x)$  instead of  $p(c|x)$ , where  $\eta_c(x)$  is the variance of the classifier given an input  $x$ .<sup>6</sup> Therefore, the obtained boundary may drift from  $x^*$  to  $x_b$ , as illustrated

<sup>6</sup>According to the bias-variance decomposition [48], the added error for the output of a classifier includes the bias of the learning algorithm and the variance. Here we focus on the variance only, since our primary goal is to find a better weight setting that can reduce the prediction error introduced by the variance.



in Figure 2.5, consequently causing prediction errors in the darkly shaded region.<sup>7</sup> This expected prediction error  $Err$  can be expressed as

$$Err = \int_{-\infty}^{\infty} A(b) f_b(b) db, \quad (2.1)$$

where  $b = x^* - x_b$ ,  $A(b)$  is the darkly shaded region, and  $f_b$  is the density function for  $b$ . Tumer and Ghosh [56] prove that (2.1) can be calculated by

$$Err = \frac{\sigma_{\eta_c}^2}{s}, \quad (2.2)$$

where  $\sigma_{\eta_c}^2$  is the variance of  $\eta_c(x)$ , and  $s$  is the difference between the derivatives of  $p(c_j|x^*)$  and  $p(c_i|x^*)$ .

Assume that DEDS includes  $k$  individual classifiers  $D_1, D_2, \dots, D_k$ , and the output of each classifier  $D_i$  is

$$f_c^i(y) = p(c|x) + \eta_c^i(x).$$

When DEDS combines these classifiers with weights  $w_i, i = 1, \dots, k$  to build an ensemble  $E$ , the output of this ensemble classifier is

$$\begin{aligned} f_c^E(y) &= \sum_{i=1}^k w_i f_c^i(y) / \sum_{i=1}^k w_i \\ &= \sum_{i=1}^k w_i p(c|x) / \sum_{i=1}^k w_i + \sum_{i=1}^k w_i \eta_c^i(x) / \sum_{i=1}^k w_i \\ &= p(c|x) + \underbrace{\sum_{i=1}^k w_i \eta_c^i(x) / \sum_{i=1}^k w_i}_{\text{variance of the ensemble classifier, i.e., } \eta_c^E(x)}. \end{aligned}$$

We can further assume that the variances of individual classifiers  $D_i, i = 1, \dots, k$  are

<sup>7</sup>The lightly shaded region is the inherent error of the Bayes optimum decision.



independent, and as derived by Wang et al. [58], the variance of  $\eta_c^E(x)$  is

$$\sigma_{\eta_c^E}^2 = \sum_{i=1}^k w_i^2 \sigma_{\eta_c^i}^2 / \left( \sum_{i=1}^k w_i \right)^2. \quad (2.3)$$



If DEDS adopts equal weighting (i.e.,  $w_i = 1/k$ ), by using (2.2) and (2.3), the expected prediction error of DEDS becomes

$$Err_{equal} = \frac{\sigma_{\eta_c^E}^2}{s} = \frac{1}{sk^2} \sum_{i=1}^k \sigma_{\eta_c^i}^2. \quad (2.4)$$

In contrast, if DEDS adopts accuracy-based weighting and sets the weights of classifiers to be inversely proportional to their error rate (i.e.,  $w_i = \alpha/\sigma_{\eta_c^i}^2$ , where  $\alpha$  is a constant), the expected prediction error of DEDS then becomes

$$Err_{acc\_based} = \frac{\sigma_{\eta_c^E}^2}{s} = \frac{1}{s} \left( 1 / \sum_{i=1}^k \frac{1}{\sigma_{\eta_c^i}^2} \right). \quad (2.5)$$

With Cauchy's Inequality, we can further derive that

$$\sum_{i=1}^k \frac{1}{\sigma_{\eta_c^i}^2} \sum_{i=1}^k \sigma_{\eta_c^i}^2 \geq \left( \sum_{i=1}^k \frac{1}{\sigma_{\eta_c^i}} \sigma_{\eta_c^i} \right)^2 = k^2,$$

which implies that

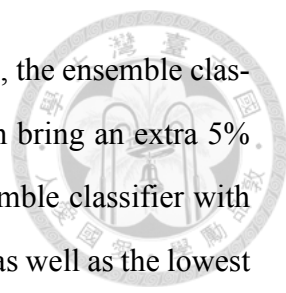
$$\frac{1}{k^2} \sum_{i=1}^k \sigma_{\eta_c^i}^2 \geq 1 / \sum_{i=1}^k \frac{1}{\sigma_{\eta_c^i}^2}.$$

By multiplying  $1/s$  on both sides of the above inequality, together with (2.4) and (2.5), we can show that

$$Err_{equal} = \frac{1}{sk^2} \sum_{i=1}^k \sigma_{\eta_c^i}^2 \geq \frac{1}{s} \left( 1 / \sum_{i=1}^k \frac{1}{\sigma_{\eta_c^i}^2} \right) = Err_{acc\_based}.$$

The theorem follows. □

The experimental results shown in Figures 2.4(a) and 2.4(b) further support this conclusion. In Figure 2.4(a), while the ensemble classifier with equal weighting (i.e., en-



sem\_EW\_WOD) already outperforms the four individual classifiers, the ensemble classifier with accuracy-based weighting (i.e., ensem\_AW\_WOD) can bring an extra 5% increase in AUC and also a 10% decrease in MSE. That is, the ensemble classifier with accuracy-based weighting tends to have the highest true positive rate as well as the lowest prediction error. The results in Figure 2.4(b) also exhibit the analogous improvements. Therefore, we consider accuracy-based weighting while evaluating the DEDS framework for the rest of this chapter.

## 2.5.2 Ensemble Size Augmentation

Rokach [42] reviews various ensemble techniques and categorizes them into two groups: dependent frameworks and independent frameworks. In a dependent framework, the output of a classifier is used as the input to construct the next classifier. In contrast, the classifiers in an independent framework are built independently, and then their outputs are combined to generate the final decision. One solid reason to design DEDS as an independent framework is exactly that every individual classifier can be trained and used to generate predictions independently. This enables DEDS to fully utilize all the CPUs or cores in order to simultaneously train and run the maximum number of individual classifiers. Since these classifiers are trained and run in parallel, the total training and prediction time will not grow much, while the prediction accuracy can be raised substantially.

To further increase the ensemble size, we need to generate more than one sparsified network from each sparsification method. For the sparsification method that has an inherent random flavor (i.e., random sparsification or random-walk-based sparsification), executing it multiple times can generate slightly varied copies of sparsified networks. The short-path-based sparsification method can produce slightly different copies of sparsified networks by modifying the length threshold  $L$ . For example, if  $L = 5$  at the beginning, using  $L = 5 \pm 1$  and  $L = 5 \pm 2$  can produce four other more sparsified networks. As for the degree-based sparsification, we can moderately disturb the original sparsified network, by replacing some existing edges in the sparsified network with the edges outside

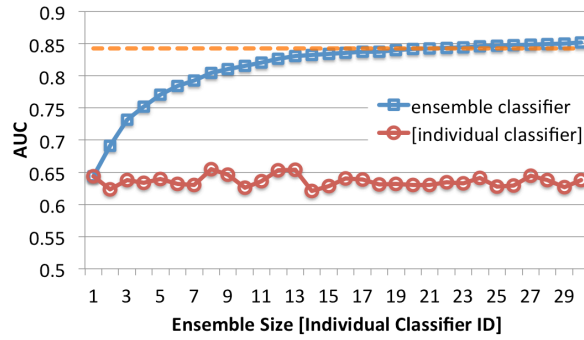


Figure 2.6: Comparison of AUC for different ensemble sizes. The dash line indicates the AUC of the classifier trained from the original network. (condmat, sparsification ratio = 15%)

the sparsified network. In order to maintain the core design behind the degree-based sparsification (i.e., preserving the edge with high summation of the degrees at its two ends), the probability of the replacement is set to be proportional to  $SD_i/SD_o$ , where  $SD_i$  and  $SD_o$  are the summation of degrees for the edges in the sparsified network and the edges outside the sparsified network, respectively.

Figure 2.6 shows the performance of ensemble classifiers with different ensemble sizes. As expected, the AUC increases as the ensemble size becomes larger. When the ensemble size becomes sufficiently large, the ensemble classifier may even slightly outperform the classifier trained from the original network. However, the AUC of the ensemble classifier increases slowly when the ensemble size exceeds 13, and the AUC eventually remains at approximately 0.85. The reason for this is that, when there are already many existing classifiers, a newly joined classifier tends to possess much of the same knowledge (i.e., edges) that the existing classifiers already have.

## 2.6 Analysis on Accuracy and Efficiency

As previously mentioned, the major challenge in drastic sparsification for link prediction is how to considerably reduce the network size while keeping the accuracy high. In this section, we present a more detailed analysis of (1) how a diverse ensemble helps to preserve the prediction accuracy, and (2) how sparsification helps to reduce the processing

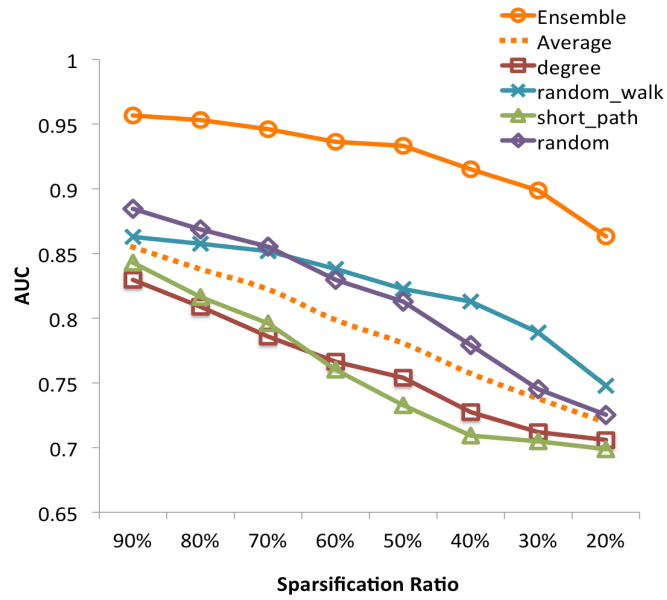
time of prediction.



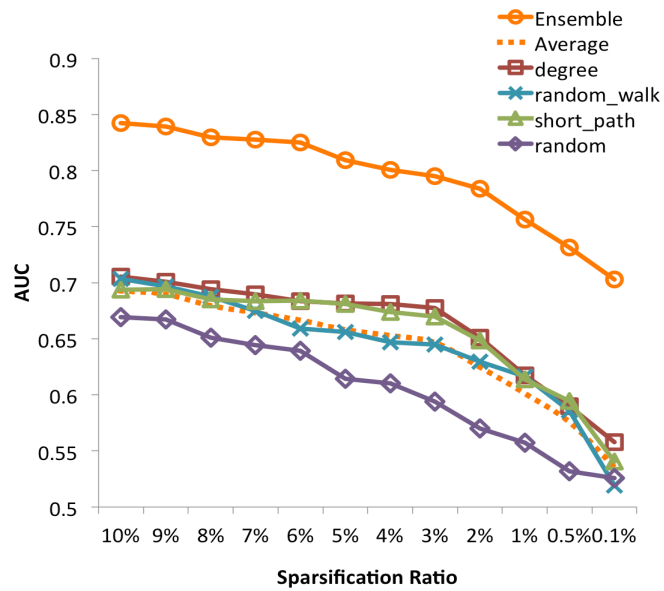
### 2.6.1 Prediction Accuracy

Here we first analyze the AUC at different sparsification ratios. The results of larger sparsification ratios (e.g., 60% or 30%) are provided in Figure 2.7(a), and then we dig into smaller sparsification ratios (e.g., 5% or 1%) in Figure 2.7(b). From Figure 2.7(a), first it can be seen that no matter what the sparsification ratio is, the AUC of the ensemble classifier is always much higher than the individual classifiers. The AUC value is 0.73 on average for individual classifiers when the sparsification ratio is 30%; in contrast, with various types of models combined together, the AUC value of the ensemble classifier can reach up to 0.90, which is a 23% improvement. When the sparsification ratio is not too small (specifically, no smaller than 40%), the ensemble classifier can even outperform the classifier trained from the original network. As intuition suggests, the prediction accuracy drops as the sparsification ratio goes down. In Figure 2.7(b), the AUC of an individual classifier trained from the random-walk-based sparsified network falls to 0.52 when the sparsification ratio is 0.1%, which is almost no different from making decisions by flipping a coin. In contrast, the DEES framework can effectively relieve the loss of prediction accuracy, and the AUC value of the ensemble classifier is able to remain at 0.70, which is a 35% improvement.

In addition to low prediction accuracy, another problem with using only one type of sparsified network is that the performance of individual classifiers is unstable, and the ranking of these individual classifiers varies among different sparsification ratios. For example, random sparsification is the most effective method when the sparsification ratio is larger than 70%, but the short-path-based sparsification comes out on top when the sparsification ratio is 60%. The ranking then varies again at the 9% sparsification ratio, where the degree-based sparsification outperforms the other three methods. Therefore, it is not possible here to choose an individual classifier that has top performance for all sparsification ratios. This difficulty, once again, shows the importance of incorporating the



(a) disease, ensemble size = 5



(b) disease, ensemble size = 30

Figure 2.7: Performance of four different individual classifiers, their average (denoted as "Average"), and the ensemble classifier at different sparsification ratios.

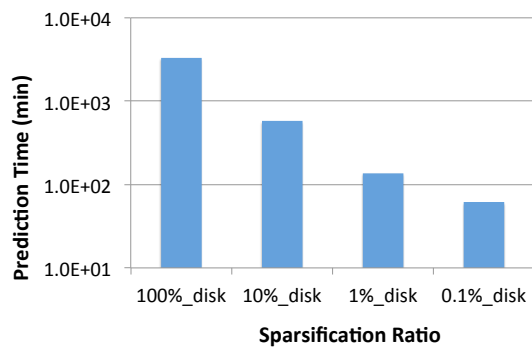
ensemble methodology, which enables the DEDS framework to have a steadily increasing AUC value and outperform every individual classifier at all sparsification ratios.

The ensemble size in Figure 2.7(b) is 30, which appears to be large. However, when the sparsification ratio is 0.1%, 30 such sparsified networks actually contain only 3% of the edges in the original network. Moreover, as shown in Figure 2.7(b), the ensemble classifier combining 30 individual classifiers trained from various 0.1% sparsified networks can outperform any type of individual classifier trained from a 3% sparsified network. Remember that all of the individual classifiers are unentangled in the DEDS framework. That is, with enough CPUs or cores, these 30 classifiers can be trained and used to generate predictions at the same time. With only the running time of a 0.1% sparsified network, the DEDS framework can provide prediction accuracy better than that of using a 3% sparsified network. These results suggest that the DEDS framework could be considered as a method for parallelizing link prediction tasks to fully utilize the computational resources.

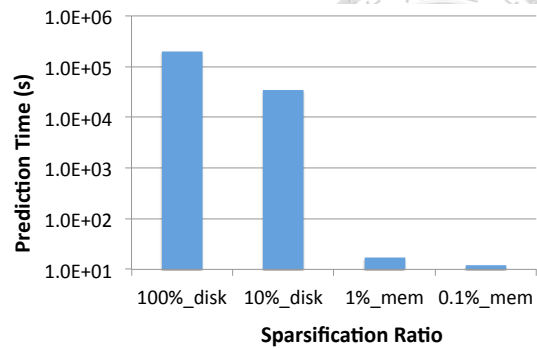
## 2.6.2 Computational Efficiency

Figure 2.8(a) shows that link prediction tasks would take more than 55 hours when using the large rmat network. In contrast, the DEDS framework only requires 4% of prediction cost if the network is sparsified down to 1% of the original one, which means the prediction time can be substantially shortened, from two days to two hours. Note that when the sparsification ratio is small, the sparsified network may be able to fit into main memory. Assume that the network used in Figure 2.8(a) can fit into main memory when the sparsification ratio is 1%. According to the experimental results in Figure 2.8(b), the prediction cost is now drastically lowered, from two days to only 17 seconds, which means that DEDS saves more than 99.94% on running time. The reason why DEDS can provide such great savings is that it not only reduces the number of edges needed to be processed, but also relieves the burden of disk access.

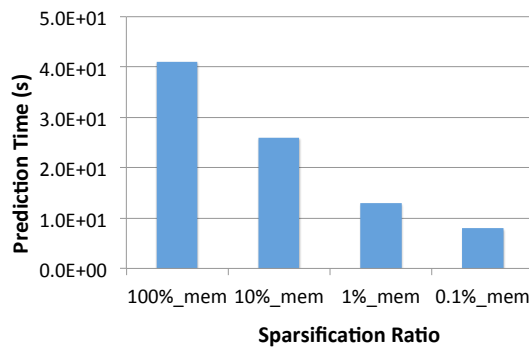
If the original network is already able to fit into main memory, the proposed DEDS framework can still lower the prediction cost considerably, by reducing the number of



(a) rmat



(b) rmat



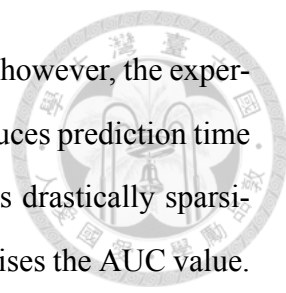
(c) condmat

Figure 2.8: Running time under different sparsification ratios: (a) the network is stored in the disk, (b) the original network is stored in the disk but can fit into main memory after being sparsified, and (c) the network is cached in main memory.

edges that need to be processed. Here we use the smaller `condmat` network and put it into main memory from the start. The experimental results in Figure 2.8(c) show that when the network is sparsified to 10% of the original one, the link prediction process requires less than 64% of the running time. If the sparsification ratio further decreases to 1% or even 0.1%, the link prediction process will only require 30% or 19% of the running time, respectively. That is, every time when the number of edges is sparsified to 10% of what it previously was, the prediction cost can be reduced by 35-50%.

## 2.7 Summary

To the best of our knowledge, there is no existing work in the literature that addresses the ensemble of drastic sparsification to facilitate link prediction. Removing some edges



before predicting the existence of other edges seems counterintuitive; however, the experimental results show that the proposed DEDS framework not only reduces prediction time considerably, but also preserves high accuracy. When the network is drastically sparsified, DEDS effectively relieves the drop in prediction accuracy and raises the AUC value. With a larger sparsification ratio, DEDS can even outperform the classifier trained from the original network. In terms of efficiency, the prediction cost is substantially reduced after the network is sparsified. If the original network is disk-resident but can fit into main memory after being sparsified, the improvement is even more significant. Moreover, the DEDS framework places no restrictions on the sparsification ratio and ensemble size, and users can decide the ratio and the size flexibly based on their computational ability and accuracy requirements. Note that all of the individual classifiers in DEDS can be trained and run independently, which makes DEDS easier to parallelize.





## Chapter 3

# Efficient Social-Temporal Group Query with Acquaintance Constraint

### 3.1 Introduction

In Chapter 2, we have studied the link prediction problem that focuses on predicting the existence of link between particular two users. In this chapter, we extend the scope from the relationship between two users to the relationship among a group of users, and study the social-temporal group query problem with its applications in activity planning. Three essential criteria for social activity planning are (1) finding a group of 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. 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<sup>1</sup>, Timebridge<sup>2</sup>,

---

<sup>1</sup>The Doodle website. <http://doodle.com/>.

<sup>2</sup>The Timebridge website. <http://www.timebridge.com/>.

and Meetup<sup>3</sup>, 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 formulate a new query problem, named *Social-Temporal Group Query (STGQ)*, which considers the available time and relationships of candidate attendees. Given an activity initiator, we consider her social network for candidate attendees. We assume that their schedules are available to the planning service (e.g., via web collaboration tools), and the closeness between friends is quantitatively captured as *social distance* [3, 10, 65]. Based on the criteria mentioned earlier, an STGQ comes with the following parameters, (1) a group size  $p$  to specify the number of attendees, (2) an availability constraint  $m$  to specify the length of activity period, (3) a social radius constraint  $s$  for the scope of candidate attendees, and (4) an acquaintance constraint  $k$  to govern the relationships between attendees. STGQ aims to find a group and time matching the group size in (1) and the activity length in (2), such that the total social distance between the initiator and attendees is minimized. Additionally, the social radius constraint in (3) specifies that all attendees are located no more than  $s$  edges away from the initiator, while the acquaintance constraint in (4) 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 and time are returned. 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 the following, Example 3.1.1 presents an illustrative example of SGTQ.

**Example 3.1.1.** Consider an illustrative example in Figure 3.1, where Casey Affleck would like to invite some friends to dine together. Figure 3.1(a) shows a possible social network of Casey Affleck. Assume that Casey Affleck is trying to find three mutually ac-

---

<sup>3</sup>The Meetup website. <http://www.meetup.com/>.

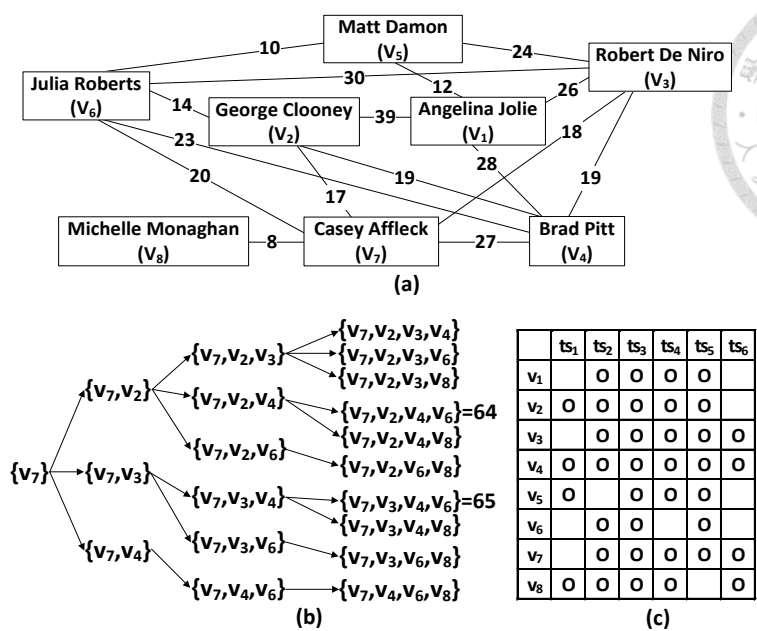
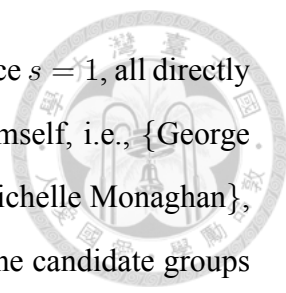


Figure 3.1: An illustrative example for STGQ. (a) The sample social network, (b) the dendrogram of candidate group enumeration and (c) schedules of candidate attendees.

quainted friends. Issuing an STGQ with  $p = 4$  and  $s = 1$ , which returns  $\{\text{George Clooney } (v_2), \text{Robert De Niro } (v_3), \text{Casey Affleck } (v_7), \text{Michelle Monaghan } (v_8)\}$ , does not ensure social closeness since the three close friends of Casey Affleck are not acquainted to each other (as shown in Figure 3.1(a)). Instead, by adding the acquaintance constraint and issuing an STGQ with  $p = 4$ ,  $s = 1$  and  $k = 1$ , a better list of invitees  $\{\text{George Clooney } (v_2), \text{Robert De Niro } (v_3), \text{Julia Roberts } (v_6), \text{Casey Affleck } (v_7)\}$ , where everyone knows at least two invitees, is obtained. However, Casey Affleck then finds out that these four attendees have no available time in common when he expects the length of activity time as 3, i.e., three consecutive time slots. Figure 3.1(c) shows the schedule of candidates, with their available time slots marked by circles. Therefore, he turns to issue an STGQ with  $p = 4$ ,  $s = 1$ ,  $k = 1$  and  $m = 3$ , which returns a group of mutually acquainted invitees and a suitable activity period that is available for all invitees, i.e.,  $\{\text{George Clooney } (v_2), \text{Robert De Niro } (v_3), \text{Brad Pitt } (v_4), \text{Casey Affleck } (v_7)\}$  and  $[ts_2, ts_4]$ .

An intuitive approach to find the answer is to enumerate and examine all the possible four-person candidate groups that include Casey Affleck himself for each possible activity period of length 3. In this example, the time interval  $[ts_1, ts_6]$  can be divided into four can-



candidate activity periods,  $[ts_1, ts_3]$ ,  $[ts_2, ts_4]$ ,  $[ts_3, ts_5]$  and  $[ts_4, ts_6]$ . Since  $s = 1$ , all directly connected friends of Casey Affleck, together with Casey Affleck himself, i.e., {George Clooney, Robert De Niro, Brad Pitt, Julia Roberts, Casey Affleck, Michelle Monaghan}, are candidates. Figure 3.1(b) illustrates the enumeration process of the candidate groups in accordance with the increasing order of user IDs. Note that the numbers 64 and 65 indicate the total social distances of qualified candidate groups. These ten non-duplicate candidate groups constitute the solution space in a certain activity period, from which we eliminate the ones disqualified by the acquaintance constraint or not available in this activity period. Finally, the group with the smallest total social distance among qualified candidate groups is returned as the optimal solution. ■

In situations where the activity time is pre-determined (e.g., a tennis game), STGQ can be simplified as a *Social Group Query (SGQ)*. In this chapter, 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., [6, 18, 43, 55, 59]), STGSelect can process both temporal and social dimensions ef-

fectively and efficiently.

Contributions of this chapter 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. In other words, 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.
- 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 less coordination effort, increasing users' willingness to organize activities.

The rest of this chapter is summarized as follows. In Section 3.2, we introduce related works. Section 3.3 formulates SGQ and explains the details of Algorithm SGSelect. Section 3.4 extends our study on SGQ to the more complex STGQ. The details of Algorithm STGSelect are also included. Finally, we present the experimental results in Section 3.5 and summarize this chapter in Section 3.6.



## 3.2 Related Works

Though some web applications have been developed to support activity coordination, they require users to manually assign 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 event planning websites and apps, such as Doodle, Timebridge, SelectTheDate<sup>4</sup>, and NeedToMeet<sup>5</sup>, are developed to reduce the initiator's efforts on collecting the available time of potential participants. However, the initiator still needs to manually choose some possible activity times and the participants to issue the invitations, and social cohesiveness is thereby not ensured. Such manual activity coordination process is tedious and time-consuming. In contrast, the proposed STGQ, complementary to the above web applications, can automatically find a group of close friends to get together at a suitable activity time.

By minimizing the total social distance among the attendees, we are actually forming a cohesive subgroup in the social network. In the field of social network analysis, research on finding various kinds of subgroups, such as clique, k-plex and k-truss has been conducted (e.g., [6, 18, 43, 55, 59]). There are some related works on group formation (e.g., [3, 44, 57]), team formation (e.g., [2, 24, 41]), and group query (e.g., [27, 28, 60]). There are also some related works on community search and social circle discovering (e.g., [35, 52, 61]). While these works focus on different scenarios and aims, none of them simultaneously encompass the social and temporal objectives to facilitate automatic activity planning. Therefore, the STGQ problem is not addressed previously.

## 3.3 Social Group Query

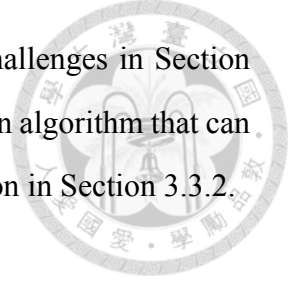
In this section, we present *Social Group Query (SGQ)*, to find the optimal group of attendees satisfying the social radius constraint and the acquaintance constraint. In the

---

<sup>4</sup>The SelectTheDate website. <http://www.selectthedata.com/>.

<sup>5</sup>The NeedToMeet website. <http://www.needtomeet.com/>.

following, we first present the problem formulation and research challenges in Section 3.3.1, and then prove that SGQ is NP-hard. Afterwards, we propose an algorithm that can effectively prune redundant search space to obtain the optimal solution in Section 3.3.2.



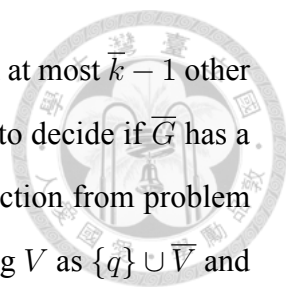
### 3.3.1 Problem Definition

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 an activity size,  $s$  is a social radius 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 different kinds of activities. For example, the initiator can ensure that all invited attendees are directly acquainted with her by specifying  $s = 1$ . On the other hand, for a party, the initiator can specify a larger  $s$  such that some friends of friends can also be invited to the party. Moreover, the initiator can vary  $k$  for different kinds of activities, e.g., a small  $k$  for a gathering where all the attendees know each other very well, while a larger  $k$  for a more diverse event. Note that the size of solution space, i.e., the number of candidate groups, rapidly grows when  $p$  and  $s$  increase. On the other hand,  $k$  serves as a filter to determine whether each candidate group satisfies the acquaintance constraint. Indeed, processing SGQ is an NP-hard problem (as proved below in Theorem 3.3.1). Fortunately, while the problem is very challenging, it is still tractable when the size of  $s$  and  $p$  are reasonable. In Section 3.3.2, we design an efficient query processing algorithm by pruning unqualified candidate groups.

**Theorem 3.3.1.** *SGQ is NP-hard and inapproximable within any ratio.*

*Proof.* In the following, we first prove that SGQ is NP-hard with a reduction from problem  $\bar{k}$ -plex, which is NP-hard [4]. A  $\bar{k}$ -plex with  $c$  vertices is a subgraph in a graph  $\bar{G} =$

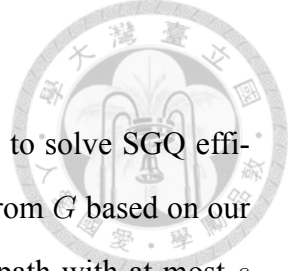


$(\bar{V}, \bar{E})$ , such that every vertex in the subgraph can share no edge with at most  $\bar{k} - 1$  other vertices in the subgraph. Problem  $\bar{k}$ -plex with parameters  $c$  and  $\bar{k}$  is to decide if  $\bar{G}$  has a  $\bar{k}$ -plex with  $c$  vertices. We prove that SGQ is NP-hard with the reduction from problem  $\bar{k}$ -plex. Specifically, we construct a weighted graph  $G(V, E)$  by letting  $V$  as  $\{q\} \cup \bar{V}$  and letting  $E$  as  $\bar{E}$ . We then add  $c$  edges into  $E$  to make  $q$  connect to all the other vertices. The distance of every edge is 1. In the following, we prove that there exists a feasible group  $F$  with  $c + 1$  attendees in  $G$  for the SGQ with  $s = 1$  and  $k = \bar{k} - 1$ , if and only if there exists a  $\bar{k}$ -plex with  $c$  vertices in  $\bar{G}$ .

We first prove the sufficient condition. If we remove  $q$  from  $F$ , every vertex in  $F$  still shares no edge with at most  $\bar{k} - 1$  other vertices, since  $q$  connects to all the other vertices. In other words,  $F - \{q\}$  with  $c$  vertices corresponds to a  $\bar{k}$ -plex with size  $c$  in  $\bar{G}$ . We then prove the necessary condition. If there exists a  $\bar{k}$ -plex with size  $c$  in  $\bar{G}$ ,  $F$  with  $q$  and the  $c$  vertices must all satisfy the social radius constraint with  $s$  as 1, since all the  $c$  vertices connect to  $q$  in  $G$ . Moreover, since each vertex in the  $\bar{k}$ -plex shares no edge with at most  $\bar{k} - 1$  other vertices,  $F$  with  $q$  and the  $c$  vertices must satisfy the acquaintance constraint.

After proving that SGQ is NP-hard, we then prove that there is no approximation algorithm for SGQ unless  $P = NP$ . Note that SGQ will return the objective value as  $\infty$  if  $F = \emptyset$ , i.e., no feasible solution exists for SGQ. Therefore, if SGQ has a polynomial-time approximation algorithm with an arbitrarily large ratio  $\rho < \infty$ , the above proof indicates that (1) the algorithm can find a feasible solution for SGQ if  $\bar{k}$ -plex returns TRUE, and (2) any SGQ instance with the algorithm returning a feasible solution implies that the corresponding instance in  $\bar{k}$ -plex is TRUE. In other words, the  $\rho$ -approximation algorithm can solve  $\bar{k}$ -plex in polynomial time, implying that  $P = NP$ , which is widely deemed incorrect. Therefore, SGQ has no polynomial-time approximation algorithm. The theorem follows. □





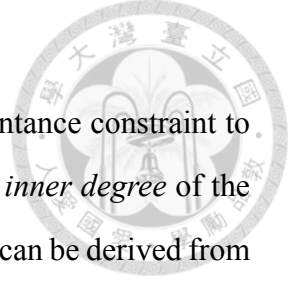
### 3.3.2 Algorithm Design

In this section, we propose a novel algorithm, namely SGSelect, to solve SGQ efficiently. Our idea is to first derive a *feasible graph*  $G_F = (V_F, E_F)$  from  $G$  based on our observation on the social radius constraint, such that there exists a path with at most  $s$  edges from  $q$  to each vertex in  $G_F$ . Starting from  $q$ , we 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., following the acquaintance constraint) to move this vertex to  $V_S$  at each iteration, until  $V_S$  has  $p$  vertices and the process stops.

The selection of a vertex from  $V_A$  at each iteration is critical to the performance of query processing. It is essential to avoid choosing a vertex  $v$  that may significantly increase the total social distance or lead to the violation of the acquaintance constraint. We observe that the access order of nodes in constructing candidate groups is a key factor to the overall performance. It is important to take a priority to consider nodes that are very likely to be included in the final answer group, i.e., the optimal solution. This may facilitate effective early pruning of unqualified solutions. Additionally, social radius and acquaintance constraints can be exploited to facilitate efficient pruning of vertices which would not lead to the eventual answer. We summarize our ideas as follows.

**Access ordering.** To guide an efficient exploration of the solution space, we access vertices in  $V_A$  following an order that incorporates (1) the increment of the total social distance and (2) the feasibility for the acquaintance constraint. Accordingly, we define the notion of *interior unfamiliarity* and *exterior expansibility* of  $V_S$  to test the feasibility of examined vertices to the acquaintance constraint during the vertex selection.

**Distance pruning.** To avoid exploring vertices in  $V_A$  that do not lead to a better solution in terms of total social distance, Algorithm SGSelect keeps track of the best feasible solution obtained so far and leverages its total social distance to prune redundant exami-



nations of certain search space.

**Acquaintance pruning.** We explore the properties of the acquaintance constraint to facilitate search space pruning. Specifically, we define the notion of *inner degree* of the vertices in  $V_A$  and derive its lower bound, such that a feasible solution can be derived from vertices in  $V_S$  and  $V_A$ . The lower bound is designed to detect the stop condition when there exists no feasible solution after including any vertex in  $V_A$ .

To find the optimal solution, Algorithm SGSelect may incur an exponential time in query processing because SGQ is NP-hard. In the worst case, all candidate groups may need to be considered. However, by employing the above pruning strategies, the average running time of Algorithm SGSelect can be effectively reduced, as to be shown in Section 3.5. In the following, we present the details of the proposed algorithm.

### Radius Graph Extraction

Obviously, the social radius constraint can effectively prune redundant candidates in the social network of the activity initiator. Thus, Algorithm SGSelect first extracts the vertices that satisfy the social radius constraint. A simple approach to meet the social radius constraint 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 those vertices that have their minimum-edge paths longer than  $s$  edges. Nevertheless, the minimum-distance path with at most  $s$  edges and the minimum-edge path can be different. As a result, the total distance of the minimum-edge path may not be the minimum distance. Moreover, the minimum-distance path may consist of more than  $s$  edges which does not satisfy the social radius constraint. To address the above problem, 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 as follows.

**Definition 3.3.1.** *The  $i$ -edge minimum distance between the vertex  $v$  and the vertex  $q$  is*

$$d_{v,q}^i = \min_{u \in N_v} \{d_{v,q}^{i-1}, d_{u,q}^{i-1} + c_{u,v}\},$$

where  $N_v$  is the set of neighboring vertices of  $v$ .

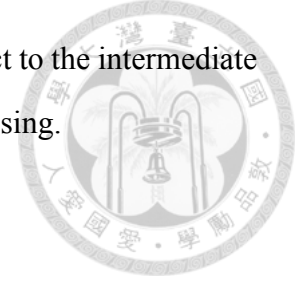
Based on dynamic programming, Algorithm SGSelect computes the  $i$ -edge minimum distance between the vertex  $v$  and the vertex  $q$  by iteratively deriving  $d_{v,q}^i$  in terms of  $d_{u,q}^{i-1}$  of each neighboring vertex  $u$ , for  $1 \leq i \leq s$ . Initially, we set  $d_{v,q}^0$  as  $\infty$  for every vertex  $v, v \neq q$ . We set  $d_{q,q}^0$  as 0 and derive  $d_{v,q}^1$  for every vertex  $v$  in  $N_q$ . At the next iteration, we update  $d_{v,q}^2$  for  $v$  if there exists a neighbor  $u$  of  $v$  such that  $d_{u,q}^1 + c_{u,v}$  is smaller than  $d_{v,q}^1$ . This case indicates that there is an alternate path from  $v$  to  $q$  via a neighbor  $u$ , and the path has a smaller total distance. Our algorithm iterates at most  $s$  times for each vertex. Therefore, each vertex  $v$  with  $d_{v,q}^s < \infty$  is extracted in our algorithm to construct a feasible graph  $G_F = (V_F, E_F)$ . In the graph, the total distance of the minimum-distance path with at most  $s$  edges (i.e.,  $d_{v,q}^s$ ) is adopted as the social distance between  $v$  and  $q$  (i.e.,  $d_{v,q}$ ). In other words, we ensure that every vertex in  $V_F$  satisfies the social radius constraint. Therefore, we consider  $G_F$  in evaluating the SGQ for the rest of this chapter.

### Access Ordering

After constructing the feasible graph  $G_F$ , Algorithm SGSelect iteratively explores  $G_F$  to find the optimal solution. Initially, the intermediate solution set  $V_S$  includes only  $q$ , and the remaining vertex set  $V_A$  is  $V_F - \{q\}$ . At each iteration afterwards, we select and move a vertex from  $V_A$  to  $V_S$  in order to expand the intermediate solution in  $V_S$ . Therefore,  $V_S$  represents a feasible solution when  $|V_S| = p$  and the vertices in  $V_S$  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 for backtracking. This process continues until  $V_S$  has  $p$  vertices.

To reduce the running time and search space, the selection of a vertex at each iteration is critical. Naturally, we would like to include a vertex that minimizes the increment of the total social distance. Nevertheless, the connectivity of the selected vertex imposes additional requirements for satisfying the acquaintance constraint. Thus, we introduce the

notion of *interior unfamiliarity* and *exterior expansibility* with respect to the intermediate solution set  $V_S$  to exploit the acquaintance constraint in query processing.



**Definition 3.3.2.** *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 set  $V_S - \{v\} - N_v$  refers to the set of non-neighboring vertices of  $v$  in  $V_S$ .

The interior unfamiliarity describes the connectivity within the intermediate solution set, and a smaller interior unfamiliarity means the vertices in  $V_S$  are more densely connected. As shown later, the interior unfamiliarity of possible intermediate solution sets are taken into account in deciding which vertex is to be included in the process of generating the candidate groups. It is preferable to first include a well-connected vertex that results in the intermediate solution set with low interior unfamiliarity since it may make selections of other vertices in the later iterations easier. Next, we define the exterior expansibility of an intermediate solution set  $V_S$ , denoted by  $A(V_S)$ , as the maximum number of vertices that  $V_S$  can be expanded from.

**Definition 3.3.3.** *The exterior expansibility of  $V_S$  is*

$$A(V_S) = \min_{v \in V_S} \{|V_A \cap N_v| + (k - |V_S - \{v\} - N_v|)\}, \quad (3.1)$$

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 exterior expansibility means the acquaintance constraint is easier to follow during the vertex selection. 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

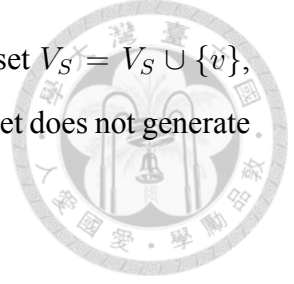
acquaintance constraint. Therefore, for a vertex  $v$  in  $V_S$ , there are at most  $|V_A \cap N_v|$  neighboring vertices and  $k - |V_S - \{v\} - N_v|$  non-neighboring vertices to be selected from  $V_A$  in order to expand  $V_S$ .

When selecting a vertex  $v$  to expand  $V_S$ , we consider both the increment of the total social distance caused by  $v$  and the connectivity of vertices in the new intermediate solution set containing  $v$ , which is captured by  $U(V_S \cup \{v\})$  and  $A(V_S \cup \{v\})$ . Specifically, Algorithm SGSelect chooses the vertex  $v$  with the minimum social distance to  $q$  that satisfies the following two conditions for interior unfamiliarity and exterior expansibility, respectively.

**Interior Unfamiliarity Condition.** The first condition considers the interior unfamiliarity. Note that a small value of interior unfamiliarity indicates that every vertex  $v \in V_S$  has plenty of neighboring vertices in  $V_S$ , i.e., the current intermediate solution set  $V_S$  is likely to be expanded into feasible solutions satisfying the acquaintance constraint. Based on this observation, we employ the interior unfamiliarity condition, i.e.,

$$U(V_S \cup \{v\}) \leq k \left[ \frac{|V_S \cup \{v\}|}{p} \right]^\theta,$$

where  $\theta \geq 0$  and  $\frac{|V_S \cup \{v\}|}{p}$  is the proportion of attendees that have been considered, to ensure the value of interior unfamiliarity remaining small when the vertex  $v$  is selected. Note that the right-hand-side (RHS) of the inequality reaches its maximum, i.e.,  $k$ , when  $\theta$  is fixed as 0. With  $\theta = 0$ , it is flexible to find 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 the vertices chosen from  $V_A$  at later iterations. Thus, the feasibility of selecting other qualified vertices in later iterations is thereby decreased. In contrast, a larger  $\theta$  allows SGSelect to choose a vertex from  $V_A$  that connects to more vertices in  $V_S$  to ensure the feasibility at later iterations. Note that the RHS of the condition increases when  $V_S$  includes more vertices. On the other hand, the algorithm reduces  $\theta$  if there exists no vertex in  $V_A$  that can satisfy the above condition. When  $\theta$  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  $V_A$  to this new intermediate solution set does not generate a feasible solution, as shown by the following lemma.

**Lemma 3.3.1.** *Given that*

$$U(V_S) > k, \tag{3.2}$$

*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  $U(V_S) > k$ , 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  $V_S$ . Please note that adding any vertex from  $V_A$  to  $V_S$  cannot 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  $v$  will remain unchanged if  $u$  and  $v$  are connected, but increase by one otherwise. Therefore, after the expansion, the number of vertices in  $V_S$  that are unacquainted with  $v$  must still exceed  $k$ . That is,  $v$  still violates the acquaintance constraint, and hence  $V_S$  cannot form a feasible solution. The lemma follows.  $\square$

**Exterior Expansibility Condition.** Now we discuss the second condition based on the exterior expansibility, which represents the maximum number of vertices in  $V_A$  that can be considered for expanding the intermediate solution set  $V_S$ , and this value 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 can satisfy 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 3.3.2.** *Given that*

$$A(V_S) < (p - |V_S|), \tag{3.3}$$

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  $A(V_S) < (p - |V_S|)$ , then we can find at least one vertex  $v$  in  $V_S$  such that  $|V_A \cap N_v| + (k - |V_S - \{v\} - N_v|) < (p - |V_S|)$ . In other words,  $(k - |V_S - \{v\} - N_v|) < (p - |V_S|) - |V_A \cap N_v|$ . As mentioned above,  $|V_S - \{v\} - N_v|$  is the number of non-neighboring vertices for  $v$ , and  $k - |V_S - \{v\} - N_v|$  thereby represents the "quota" for  $v$  to choose non-neighbor vertices from  $V_A$ . For any possible selection  $\widehat{V}_A \subseteq V_A$ , let  $\widehat{\lambda}_A$  denote the number of neighbor vertex of  $v$  in  $\widehat{V}_A$ . Since  $\widehat{\lambda}_A \leq |V_A \cap N_v|$ ,  $(p - |V_S|) - |V_A \cap N_v| \leq (p - |V_S|) - \widehat{\lambda}_A$ . Therefore, if  $A(V_S) < (p - |V_S|)$ , then  $(k - |V_S - \{v\} - N_v|) < (p - |V_S|) - \widehat{\lambda}_A$ , and  $v$  does not have enough quota to support  $\widehat{V}_A$  for satisfying the acquaintance constraint. The lemma follows.  $\square$

### Distance and Acquaintance Pruning

In the following, we further exploit two pruning strategies to reduce the 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 following distance pruning strategy avoids exploring the vertices in the remaining vertex set  $V_A$  if they do not lead to a solution with a smaller total social distance.

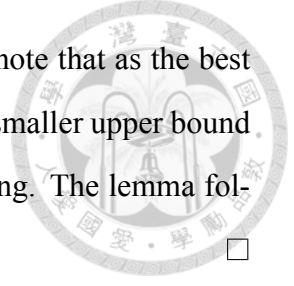
**Lemma 3.3.3.** *The distance pruning strategy stops selecting a vertex from  $V_A$  to  $V_S$  if*

$$D - \sum_{v \in V_S} d_{v,q} < (p - |V_S|) \min_{v \in V_A} d_{v,q}, \quad (3.4)$$

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, it is impossible to find an improved solution by exploring  $V_A$ , since the total social distance of any new solution must exceed  $D$  when we select  $p - |V_S|$  vertices from  $V_A$ . Algorithm SGSelect considers  $\min_{v \in V_A} d_{v,q}$  in distance pruning to avoid sorting the distances of all vertices in  $V_A$ , which requires additional com-

putation and may not be scalable for a large social network. Please note that as the best obtained solution improves at later iterations, we are able to derive a smaller upper bound in the LHS, and thus prune a larger search space with distance pruning. The lemma follows.  $\square$



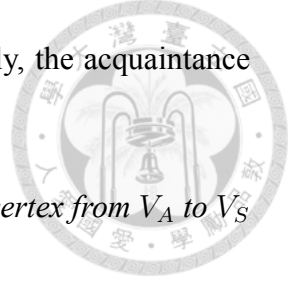
In addition to pruning the search space that does not lead to a smaller total social distance, we also propose an acquaintance pruning strategy that considers the feasibility of selecting vertices from  $V_A$ , and stops exploring  $V_A$  if there exists no solution that can satisfy the acquaintance constraint. Earlier in this section, the interior unfamiliarity and the exterior expansibility consider the connectivity between the vertices in only  $V_S$ , and the connectivity between the vertices in  $V_S$  and  $V_A$ , respectively. Here the acquaintance pruning strategy focuses on the edges between the vertices in  $V_A$ . Note that all vertices in  $V_A$  are excluded from expansion (and thus the corresponding  $V_S$  is pruned) if

$$\sum_{v \in V_A} |V_A \cap N_v| < (p - |V_S|)(p - |V_S| - k - 1).$$

The LHS of the above 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 is the lower bound on the total inner degree on any set of vertices extracted from  $V_A$  to expand  $V_S$  into a solution satisfying the acquaintance constraint. Specifically, our algorithm needs to select  $p - |V_S|$  vertices from  $V_A$ , and hence the inner degree of any selected vertex cannot be smaller than  $p - |V_S| - k - 1$ ; otherwise, the vertex must be unacquainted with more than  $k$  vertices and violate the acquaintance constraint.

The above strategy can be improved by replacing the LHS of the inequality 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. Therefore, with  $M_A$ , our algorithm is able to stop the search earlier, and prune off more infeasible solutions because  $M_A \subseteq V_A$ , and  $\sum_{v \in M_A} |V_A \cap N_v| \leq \sum_{v \in V_A} |V_A \cap N_v|$ . To obtain the exact value of  $\sum_{v \in M_A} |V_A \cap N_v|$ , we need to identify the vertices in the set  $M_A$  first, which may require sorting the vertices in  $V_A$ . However, since the size of  $M_A$  (i.e.,  $p - |V_S|$ ) is usually small, we can use a few times of extracting maximum to iden-





tify the vertices in  $M_A$ , rather than sorting the entire  $V_A$ . Specifically, the acquaintance pruning is specified as follows.

**Lemma 3.3.4.** *The acquaintance pruning strategy stops selecting a vertex from  $V_A$  to  $V_S$  if*

$$\sum_{v \in M_A} |V_A \cap N_v| < (p - |V_S|)(p - |V_S| - k - 1), \quad (3.5)$$

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

*Proof.* To acquire a tighter bound for the acquaintance pruning, we consider  $\sum_{v \in M_A} |V_A \cap N_v|$  instead of  $\sum_{v \in V_A} |V_A \cap N_v|$  in the LHS of Eq. (3.5). 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  $\sum_{v \in M_A} |V_A \cap N_v|$ , where  $M_A$  denotes the set of  $p - |V_S|$  vertices in  $V_A$  with the largest inner degrees. 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 there will be at least  $(p - |V_S|)(p - |V_S| - k - 1)$  inner degrees. When the acquaintance pruning happens, it means 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 being unacquainted with more than  $k$  vertices and violating the acquaintance constraint. Therefore, the pruned search space contains no feasible solution. The lemma follows.  $\square$

In the following, we first prove that our algorithm with the above strategies finds the optimal solution. After that, Example 3.3.1 provides illustration of Algorithm SGSelect, and the pseudo code of SGSelect can be found in Appendix B.

**Theorem 3.3.2.** *SGSelect obtains the optimal solution to SGQ.*

*Proof.* In radius graph extraction, each of the removed vertices has no path with at most  $s$  edges connecting to  $q$ , and no feasible solution thereby contains these vertices. Algorithm SGSelect includes three strategies: access ordering, distance pruning, and acquaintance pruning. For each  $V_S$ , interior unfamiliarity and exterior expansibility do not consider the

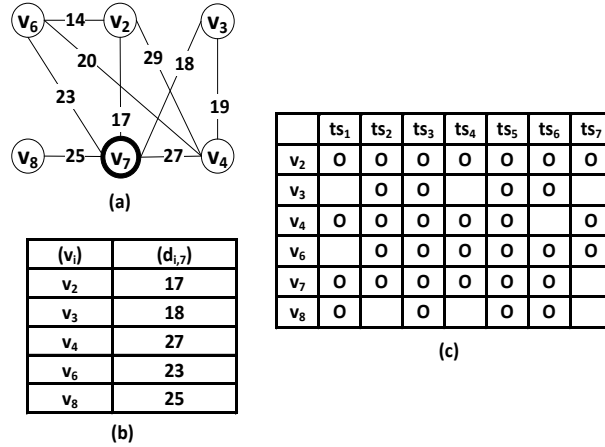


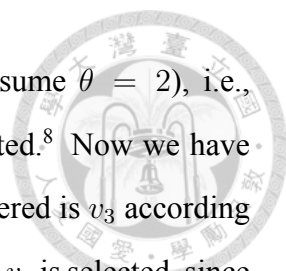
Figure 3.2: Another illustrative example for SGQ and STGQ. (a) The sample social network, (b) the social distances of candidate attendees and (c) the schedules of candidate attendees.

vertex violating the acquaintance constraint, which are proven in Lemma 3.3.1 and Lemma 3.3.2, respectively. After we choose a vertex  $u$  from  $V_A$ , Lemma 3.3.3 shows that the distance pruning specifies a lower bound on the total social distance that is derived from  $V_A$ . Therefore, the distance pruning will prune off only the solution with a larger total social distance. Moreover, Lemma 3.3.4 shows that the acquaintance pruning specifies the lower bound on the total inner degree on any set of vertices extracted from  $V_A$  in any feasible solution. If we choose the required number of vertices with the largest inner degrees from  $V_A$  and the result cannot exceed the above lower bound, the connectivity is too small for  $V_A$  to obtain a feasible solution. The theorem follows.  $\square$

**Example 3.3.1.** In this illustrating example for SGSelect, we revisit the social network in Figure 3.1(a) and assume that  $v_7$  issues an SGQ with  $p = 4$ ,  $s = 1$ , and  $k = 1$ . All candidate attendees  $v_i$  with  $d_{v_i, v_7}^1 < \infty$  are shown in Figure 3.2(a), with their social distances to  $v_7$  listed in Figure 3.2(b).<sup>6</sup> In the beginning,  $V_S = \{v_7\}$  and  $V_A = \{v_2, v_3, v_4, v_6, v_8\}$ . We first consider selecting  $v_2$  (i.e., the vertex with the smallest social distance) from  $V_A$  to expand  $V_S$ . Afterwards, we have  $A(V_S \cup \{v_2\}) = 3$  and  $(p - |V_S \cup \{v_2\}|) = 4 - 2 = 2$ , which means that the exterior expansibility condition holds if we select  $v_2$ .<sup>7</sup> In addi-

<sup>6</sup>Some small modifications are made for better illustration.

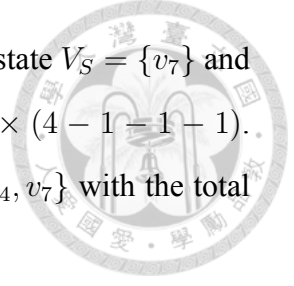
<sup>7</sup>To find  $A(V_S \cup \{v_2\})$ , we derive  $|V_A \cap N_{v_7}| + (k - |V_S - \{v_7\} - N_{v_7}|) = 4 + (1 - 0) = 5$  and  $|V_A \cap N_{v_2}| + (k - |V_S - \{v_2\} - N_{v_2}|) = 2 + (1 - 0) = 3$ , and then choose the smaller one. Therefore,  $A(V_S \cup \{v_2\}) = 3$  holds.



tion,  $U(V_S \cup \{v_2\}) = 0$  and  $k \left[ \frac{|V_S \cup \{v_2\}|}{p} \right]^\theta = 1 \times \left(\frac{2}{4}\right)^2 = \frac{1}{4}$  (assume  $\theta = 2$ ), i.e., the interior unfamiliarity condition also holds, and hence  $v_2$  is selected.<sup>8</sup> Now we have  $V_S = \{v_2, v_7\}$ ,  $V_A = \{v_3, v_4, v_6, v_8\}$ , and the next vertex to be considered is  $v_3$  according to the social distance. The exterior expansibility condition holds when  $v_3$  is selected, since  $A(V_S \cup \{v_3\}) = 1 \geq (p - |V_S \cup \{v_3\}|) = 1$ . However, it violates the interior unfamiliarity condition, since  $U(V_S \cup \{v_3\}) = 1 > k \left[ \frac{|V_S \cup \{v_3\}|}{p} \right]^\theta = 1 \times \left(\frac{3}{4}\right)^2 = \frac{9}{16}$ . We do not reduce  $\theta$  here because there are still more vertices in  $V_A$  (we put  $v_3$  in parenthesis and temporarily skip it, i.e.,  $V_A = \{(v_3), v_4, v_6, v_8\}$ ). Now the next vertex to be considered is  $v_6$  since both of the exterior expansibility condition and the interior unfamiliarity condition hold. As a result, we have  $V_S = \{v_2, v_6, v_7\}$  and  $V_A = \{v_3, v_4, v_8\}$ . Again, selecting  $v_3$  violates the interior unfamiliarity condition, so we temporarily skip  $v_3$ . When selecting  $v_8$ , we observe that it violates the exterior expansibility condition and then remove  $v_8$  from  $V_A$ . Therefore, we choose  $v_4$  instead and obtain the first feasible solution  $\{v_2, v_4, v_6, v_7\}$  (total social distance = 64). Note that if we set a small  $\theta$  and allow  $v_3$  to be selected earlier, it leads to the generation of an infeasible candidate group  $\{v_2, v_3, v_6, v_7\}$ , instead of the first feasible solution. If we can acquire the first feasible solution early, we are able to leverage the distance pruning strategy. Now SGSelect backtracks one step to the state  $V_S = \{v_2, v_6, v_7\}$  and  $V_A = \{(v_3)\}$ , and it reduces  $\theta$  since there is no other vertex in  $V_A$ . However, selecting  $v_3$  still violates the interior unfamiliarity condition when we reduce  $\theta$  to 0, since  $U(V_S \cup \{v_3\}) = 2$  and  $k \left[ \frac{|V_S \cup \{v_3\}|}{p} \right]^\theta = 1 \times \left(\frac{4}{4}\right)^0 = 1$ . Therefore, we can remove  $v_3$  and backtrack to the state  $V_S = \{v_2, v_7\}$  and  $V_A = \{(v_3), v_4, v_8\}$ . In this branch, SGSelect first removes  $v_8$  due to the violation of the exterior expansibility condition, and later we select  $v_4$  and  $v_3$  to obtain the second feasible solution  $\{v_2, v_3, v_4, v_7\}$  (total social distance = 62). Note that when reducing  $\theta$  to 0 here, the interior unfamiliarity condition holds and  $v_3$  can be selected since  $U(V_S \cup \{v_3\}) = 1$  and  $k \left[ \frac{|V_S \cup \{v_3\}|}{p} \right]^\theta = 1 \times \left(\frac{4}{4}\right)^0 = 1$ . Now SGSelect further backtracks to the state  $V_S = \{v_7\}$  and  $V_A = \{v_3, v_4, v_6, v_8\}$  and selects  $v_3$ . However, since  $62 - 18 < (4 - 2) \times 23$ , the distance pruning strategy then stops

<sup>8</sup>To calculate  $U(V_S \cup \{v_2\})$ , it is necessary to consider the value of  $|V_S - \{v_7\} - N_{v_7}|$  and  $|V_S - \{v_2\} - N_{v_2}|$  and then choose the larger one. Since  $|V_S - \{v_7\} - N_{v_7}| = |\emptyset| = 0$  and  $|V_S - \{v_2\} - N_{v_2}| = |\emptyset| = 0$ ,  $U(V_S \cup \{v_2\}) = 0$  holds.

selecting a vertex from  $V_A$ . Later, there is no need to explore the last state  $V_S = \{v_7\}$  and  $V_A = \{v_4, v_6, v_8\}$ , since  $(1 + 1 + 0) - (3 - 4 + 1) \times 0 < (4 - 1) \times (4 - 1 - 1 - 1)$ . Hence the algorithm stops and returns the optimal solution  $\{v_2, v_3, v_4, v_7\}$  with the total social distance as 62. ■



### 3.4 Social-Temporal Group Query

In the following, we first extend SGQ to STGQ by exploring the temporal dimension and formulate the problem in Section 3.4.1. STGQ is more complex than SGQ because there may exist numerous activity periods with different candidate groups. An intuitive approach is to first find the SGQ solution for each individual activity period and then select the one with the minimum total social distance. However, this approach is computationally expensive. To address this issue, in Section 3.4.2, we identify pivot time slots, the only time slots required to be explored in the temporal dimension, to facilitate efficient STGQ processing. Moreover, we propose the availability pruning strategy to leverage the correlation in the available time slots among candidate attendees to avoid exploring an unsuitable activity period.

#### 3.4.1 Problem Definition

STGQ generalizes SGQ by considering the available time of each candidate attendee via the *availability constraint*, which ensures that all selected attendees are available in a period of  $m$  time slots. 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-temporal group query  $STGQ(p, s, k, m)$ , where  $p$  is an activity size,  $s$  is a social radius constraint,  $k$  is an acquaintance constraint, and  $m$  is an activity length, 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  $t + m - 1$ .



### 3.4.2 Algorithm Design

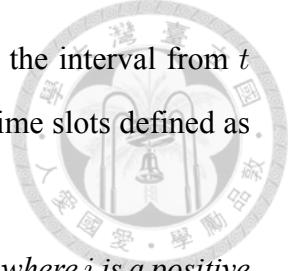
STGQ is also an NP-hard problem because STGQ can be reduced to SGQ if every candidate attendee is available in all time slots. An intuitive approach to evaluate STGQ is to consider the social dimension and the temporal dimension separately, by sequentially exploring each time slot  $t$  and the candidate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

**Pivot time slot.** We consider only a limited number of slots, namely, the *pivot time slots*, to find the solution. STGSelect returns optimal solutions even though only parts of the slots are considered.

**Access ordering.** In addition to interior unfamiliarity and exterior expansibility discussed earlier, we further consider the solution quality and the feasibility based on the availability constraint. Algorithm STGSelect constructs the  $V_S$  with vertices which have more available time slots in common to find an initial feasible solution and then chooses the vertices in  $V_A$  with smaller social distances to improve the solution.

**Availability pruning.** In addition to the distance and acquaintance pruning discussed in Section 3.3.2, we propose the availability pruning strategy to stop the algorithm when selecting any vertex from  $V_A$  never leads to a solution with  $m$  available 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 in Section 3.5, the average running time of the proposed algorithm with the above strategies can be effectively reduced, especially for a large  $m$ . In the following, we describe the details of Algorithm STGSelect, paying



special attention on the temporal dimension. Instead of considering the interval from  $t$  to  $t + m - 1$  for each time slot  $t$ , our algorithm leverages the pivot time slots defined as follows to reduce running time.

**Lemma 3.4.1.** *A time slot is a pivot time slot if the ID of the slot is  $im$ , where  $i$  is a positive integer. Any feasible solution to STGQ must include exactly one pivot time slot.*

*Proof.* If a solution does not span over a pivot time slot, the solution must have fewer than  $m$  slots because there are  $m - 1$  time slots between any two consecutive pivot time slots. If a solution contains more than one pivot time slot, the solution includes more than  $m$  slots, and the above two cases are not feasible. Moreover, there must exist an integer  $i^*$  such that the optimal solution resides in an interval starting from slot  $(i^* - 1)m + 1$  to  $(i^* + 1)m - 1$ , corresponding to pivot time slot  $i^*m$ . If the optimal solution is not located in the above interval, the optimal solution must include at least two pivot time slots and thereby is infeasible, or the optimal solution must reside in the corresponding interval for pivot time slot  $(i^* - 1)m$  or  $(i^* + 1)m$ . The lemma follows.  $\square$

**Definition 3.4.1.** *Every vertex  $v$  in the feasible graph  $G_F^{im} = (V_F^{im}, E_F^{im})$  for pivot time slot  $im$  has at least  $m$  consecutive available time slots in the interval from slot  $(i - 1)m + 1$  to  $(i + 1)m - 1$ . Moreover, there exists a path from  $q$  to  $v$  with at most  $s$  edges.*

For each pivot time slot  $im$ , Algorithm STGSelect extends SGSelect by considering the temporal information when selecting a vertex from  $V_A$  to  $V_S$ . Specifically, let  $T_S$  denote the set of consecutive time slots available for all vertices in  $V_S$ , and  $T_S$  must contain slot  $im$ . In other words,  $T_S$  will be a feasible solution to the STGQ when  $V_S$  includes  $p$  vertices satisfying the acquaintance constraint, and  $|T_S| \geq m$ . At each iteration, for each vertex in  $V_A$ , Algorithm STGSelect considers the social distance to  $q$  during the selection to reduce the objective value. However, we also consider the temporal availability of the vertex to avoid choosing a vertex that leads to a small increment of the total social distance but ends up with redundant examination of solutions eventually disqualified by the availability constraint. In other words, in addition to interior unfamiliarity and exterior expansibility as described in Section 3.3.2, we define the notion of *temporal extensibility* as follows.



**Definition 3.4.2.** *The temporal extensibility of  $V_S$  is*

$$X(V_S) = |T_S| - m.$$

*A larger temporal extensibility ensures that many vertices in  $V_A$  with good quality in the temporal dimension can be selected by our algorithm afterward.*

**Temporal Extensibility Condition.** To consider both the solution quality and feasibility in the temporal dimension, Algorithm STGSelect chooses the vertex  $u$  with the minimum social distance to  $q$ , and  $u$  must satisfy

$$X(V_S \cup \{u\}) \geq (m - 1) \left[ \frac{p - |V_S \cup \{u\}|}{p} \right]^\phi,$$

where  $\phi \geq 1$  and  $\frac{p - |V_S \cup \{u\}|}{p}$  is the proportion of attendees that have not been considered. The RHS grows when  $\phi$  decreases, and the above condition enforces that the result  $V_S \cup \{u\}$  must be more temporal extensible, i.e., more available time slots are shared by all vertices in the result, and hence more vertices in  $V_A$  are eligible to be selected at later iterations. In the extreme case, if  $\phi = 1$ , the above condition requires that the result contains almost  $2m - 1$  available time slots when  $V_S = \{q\}$ , because the RHS is close to  $m - 1$ . In contrast, as  $\phi$  grows, our algorithm is able to choose a vertex with a smaller social distance because more vertices can satisfy the above condition. Please note that  $\phi$  is increased by the algorithm if there exists no vertex in  $V_A$  that can satisfy the above condition, and the RHS approaches 0 in this case. For the case that leads to  $X(V_S \cup \{u\}) < 0$ , we remove  $u$  from  $V_A$  because adding  $u$  to  $V_S$  results in unqualified solutions that are infeasible in the temporal dimension.

In addition to distance pruning and acquaintance pruning that consider the social dimension, we propose availability pruning in the temporal dimension. The strategy enables our algorithm to stop exploring  $V_A$  if there exists no solution that can satisfy the availability constraint. The above temporal extensibility considers the available time slots for vertices in  $V_S$ . In contrast, availability pruning reduces the search space according to the available

time slots of vertices in  $V_A$ . Specifically, for each pivot time slot  $im$ , let  $\bar{t}_A^+$  and  $\bar{t}_A^-$  denote the time slots closest to  $im$ , such that all vertices in  $V_A$  are not available in the two time slots, where  $\bar{t}_A^+ > im$  and  $\bar{t}_A^- < im$ , respectively. Therefore, we are able to stop considering  $V_A$  when  $\bar{t}_A^+ - \bar{t}_A^- \leq m$ . In this case, the solution is infeasible since the interval starting from  $\bar{t}_A^- + 1$  to  $\bar{t}_A^+ - 1$  contains fewer than  $m$  time slots. This strategy can be further improved by considering the number of vertices that are not available for each time slot, and the availability pruning strategy is formally specified as follows.

**Lemma 3.4.2.** *The availability pruning strategy stops selecting a vertex from  $V_A$  to  $V_S$  if*

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

where  $\bar{t}_A^+(n)$  and  $\bar{t}_A^-(n)$  denote the time slots closest to  $im$ , such that at least  $n$  vertices in  $V_A$  are not available, and  $\bar{t}_A^+(n) > im$  and  $\bar{t}_A^-(n) < im$ , respectively. Moreover, the availability pruning strategy can prune the search space with no feasible solution.

*Proof.* If the above condition holds, there are at most  $p - |V_S| - 1$  vertices of  $V_A$  available in each of the above two slots, and we can never find a feasible solution 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. The lemma follows.  $\square$

**Theorem 3.4.1.** *STGSelect obtains the optimal solution to STGQ.*

*Proof.* Each pivot time slot is separated from a neighbor pivot time slot with  $m - 1$  time slots. Therefore, Lemma 3.4.1 shows that any feasible solution must include exactly one pivot time slot. In addition, the proposed algorithm considers the interval with  $2m - 1$  slots for each pivot time slot, and we derive the best solution by extending Algorithm SGSelect with the temporal extensibility. Moreover, Lemma 3.4.2 shows that the availability pruning discards  $V_A$  only when there exists no feasible solution satisfying the availability constraint by incorporating any vertex from  $V_A$ . The solution obtained by Algorithm STGSelect is optimal because the algorithm chooses the pivot time slot and the corresponding group with the smallest total social distance at the end of the algorithm. The



theorem follows.  $\square$

In the following, Example 3.4.1 provides illustration of Algorithm STGSelect, and the pseudo code of STGSelect can be found in Appendix B.



**Example 3.4.1.** In this illustrating example for STGSelect, we extend the SGQ in Example 3.3.1 by considering the length of activity time as 3 (i.e.,  $m = 3$ ). When processing an STGQ, the schedules of candidate attendees provided in Figure 3.2(c) should be considered as well. Since  $m = 3$ ,  $ts_3$  and  $ts_6$  are selected to be pivot time slots. For the first pivot time slot  $ts_3$ ,  $V_S = \{v_7\}$  and  $V_A = \{v_2, v_3, v_4, v_6, v_8\}$  in the beginning. As obtained in Example 3.3.1, both of the exterior expansibility condition and the interior unfamiliarity condition hold when selecting  $v_2$ . Note that STGSelect also evaluates the temporal extensibility condition when selecting a vertex to ensure the feasibility in the temporal dimension. Since  $(m - 1) \left[ \frac{p - |V_S \cup \{v_2\}|}{p} \right]^\phi = 2 \times \left(\frac{2}{4}\right)^2 = \frac{1}{2}$  (assume  $\phi = 2$ ) and  $X(V_S \cup \{v_2\}) = 2^9$ , the temporal extensibility condition also holds, and hence we can select  $v_2$  from  $V_A$  to  $V_S$ . Now we have  $V_S = \{v_2, v_7\}$  and  $V_A = \{v_3, v_4, v_6, v_8\}$ . The later vertex selection ordering is identical to Example 3.3.1 since there is no violation on the temporal constraint, and we also obtain the first feasible solution  $\{v_2, v_4, v_6, v_7\}$  (total social distance = 64) available in the activity period  $[ts_2, ts_4]$ . Until we select  $v_3$  in the state  $V_S = \{v_2, v_4, v_7\}$  and  $V_A = \{v_3\}$ , we find out that the temporal extensibility condition does not hold when selecting  $v_3$ , and then we increase  $\phi$  since there is no other vertex in  $V_A$  that we can choose. However, since  $X(V_S \cup \{v_3\}) = 2 - 3 = -1$ , the temporal extensibility condition does not hold even when the RHS of the inequality approaches 0. Therefore, we can remove  $v_3$  and backtrack to the state  $V_S = \{v_7\}$  and  $V_A = \{v_3, v_4, v_6, v_8\}$ . As shown in Example 3.3.1, the later branches violate the social constraints and hence lead to no feasible group. Therefore,  $\{v_2, v_4, v_6, v_7\}$  is the only feasible group available in activity periods extended from the pivot time slot  $ts_3$ .

Next, we start processing the second pivot time slot, i.e.,  $ts_6$ . Different from  $ts_3$ , we have  $V_S = \{v_7\}$  and  $V_A = \{v_2, v_3, v_6, v_8\}$  in the beginning. Since  $v_4$  is not available in the

---

<sup>9</sup> $T_s = \{ts_1, ts_2, ts_3, ts_4, ts_5\}$  since  $v_2$  and  $v_7$  are available in them. Hence  $|T_s| = 5$  and  $X(V_S \cup \{v_2\}) = 5 - 3 = 2$ .

pivot time slot, we can directly remove it without further consideration. We then obtain  $V_S = \{v_2, v_7\}$  and  $V_A = \{v_3, v_6, v_8\}$  because selecting  $v_2$  violates no constraint. Note that the LHS of the availability pruning condition is  $\bar{t}_A^+(|V_A| - p + |V_S| + 1) - \bar{t}_A^- (|V_A| - p + |V_S| + 1) = \bar{t}_A^+(3 - 4 + 2 + 1) - \bar{t}_A^-(3 - 4 + 2 + 1) = \bar{t}_A^+(2) - \bar{t}_A^-(2)$ . Since there are 2 vertices, i.e.,  $v_3$  and  $v_8$ , in  $V_A$  not available in  $ts_4$ ,  $\bar{t}_A^-(2) = 4$ . Besides, there are 2 vertices, i.e.,  $v_3$  and  $v_8$ , in  $V_A$  not available in  $ts_7$ ,  $\bar{t}_A^+(2) = 7$ . Therefore, the availability pruning condition holds since  $\bar{t}_A^+(2) - \bar{t}_A^-(2) = 7 - 4 \leq m$ , and we can stop selecting vertices from  $V_A$  to  $V_S$ . Then we backtrack one step to the state  $V_S = \{v_7\}$  and  $V_A = \{v_3, v_6, v_8\}$ . We can skip this final branch since the acquaintance pruning condition holds. Therefore, there exists no feasible group available in activity periods extended from  $ts_6$ . Finally, we return the group  $\{v_2, v_4, v_6, v_7\}$  and the time period  $[ts_2, ts_4]$  as the optimal result. ■

## 3.5 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 3.5.1. We then 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 3.5.2. Finally, we invite users to answer activity planning problems and compare the proposed algorithm with manual coordination in Section 3.5.3.

### 3.5.1 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., [2,27,41]) 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 [12] (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 [2,27,41] and correlated to [11,39,55].

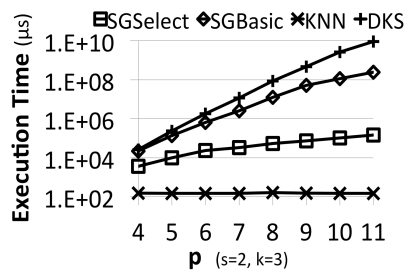
Since evaluating STGQ requires daily schedules, we invite 194 people from various community to join the experiment and use Google Calendar to collect their schedules. The data collection lasts for 5 weeks and returns 6790 days of real schedules. We then randomly select 5 weekday and 2 weekend schedules to form a 7-day one for each vertex in the social network. We perform a series of sensitivity tests on a coauthorship network (called Coauthor) of 16,726 people [38]. In addition to the sensitivity tests, we also conduct experiments on a much larger YouTube social network [61], which includes 1,134,890 people. The algorithms are implemented on an HP DL580 server with four Intel E7-4870 2.4 GHz CPUs and 128 GB RAM.

To demonstrate the strength of automatic group recommendation, we implement a social activity planning application on Facebook and conduct a user study. We invite 171 people from various communities to perform manual activity planning. Each user answers 20 activity planning requests with social graphs extracted from their social networks on Facebook. The social distances between users and their friends are specified by the users, and the social distances between their friends are derived according to the number of common friends [9, 37]. These 20 tasks span various network sizes and different numbers of attendees, and with different  $s$  and  $k$  for varying social atmospheres. We then compare the solution quality and the processing time of manual coordination with that of automatic group recommendation.

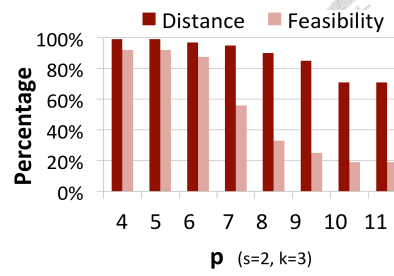
### 3.5.2 Performance Analysis of SGQ and STGQ

In this section, we first present an analysis on the proposed strategies in SGQ and its extension in temporal dimension (i.e., STGQ). After that, we then evaluate the proposed algorithms in the large YouTube dataset.

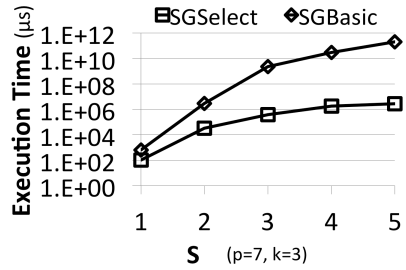
**Analysis of SGQ.** We first compare the running time of SGSelect against SGBasic,



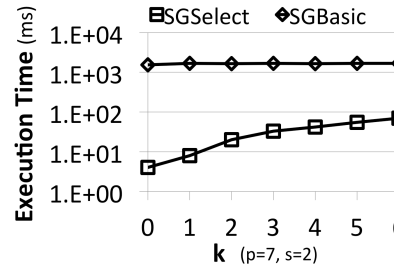
(a) Comparison of running time with different  $p$ .



(b) Solution quality analysis of KNN.



(c) Comparison of running time with different  $s$ .

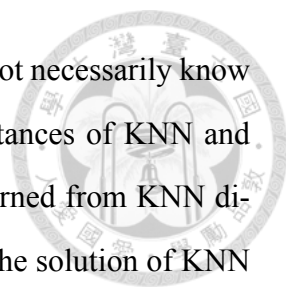


(d) Comparison of running time with different  $k$ .

Figure 3.3: Experimental results of SGQ.

KNN, and DKS with different numbers of attendees, i.e.,  $p$ . Figure 3.3(a) presents the experimental results with  $s = 2$  and  $k = 3$ . The trends in other parameter settings, such as  $s = 1$ , are similar. The results indicate that SGSelect outperforms SGBasic and DKS, and the improvement becomes more significant as  $p$  grows because SGBasic and DKS need to carefully examine numerous candidate groups, and the processing effort of each candidate group also increases with  $p$ . SGBasic outperforms DKS for a larger  $p$  because the constraint with the same  $k$  becomes stricter under a larger  $p$ . Moreover, SGBasic is likely to detect and then discard an infeasible candidate group in an early stage (i.e., find a candidate group infeasible when checking the first one or two vertices instead of checking all the vertices in the group). However, DKS only focuses on maximizing the density of the candidate group and does not leverage the  $k$  constraint. In contrast to the above approaches, SGSelect is able to effectively prune the solution space with the proposed access ordering, distance pruning, and acquaintance pruning strategies.

Although KNN is the fastest one in Figure 3.3(a), Figure 3.3(b) manifests that many solutions returned by KNN are infeasible to SGQ. More specifically, the *feasibility* ratio shows the percentage of feasible groups returned by KNN. It drops quickly as  $p$  grows

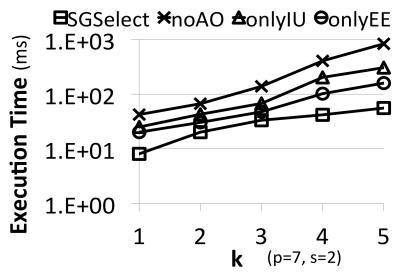


because the candidates with small social distances to the initiator do not necessarily know each other. In addition, Figure 3.3(b) compares the total social distances of KNN and SGSelect. The *distance* ratio represents the total social distance returned from KNN divided by the total social distance returned from SGSelect. Note that the solution of KNN can be regarded as a lower bound on the total social distance of SGQ since the social constraint is relaxed in KNN. Figure 3.3(b) indicates that the ratio remains above 70% even for a large  $p$ .

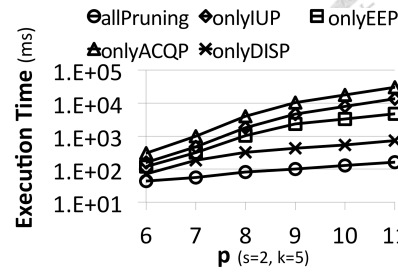
Figure 3.3(c) shows the results with different social radius constraints, i.e.,  $s$ . As  $s$  rises, the number of candidate vertices considered (i.e., friends within  $s$  hops) increases quickly, and the running time grows rapidly as a consequence. For example, when  $s$  changes from 2 to 3, the running time of SGBasic drastically becomes near 1,000 times greater. However, the running time of SGSelect only increases 11 times. This result indicates that the proposed pruning strategies become more and more effective as the number of candidates increases. SGSelect thereby is much more scalable than SGBasic. In addition to the social radius constraint, we also compare the running time of these two approaches under different acquaintance constraints, i.e.,  $k$ . As shown in Figure 3.3(d), the running time of SGBasic only slightly changes for different  $k$ . In contrast, the running time of SGSelect is reduced for a smaller  $k$ , since the IU pruning, EE pruning and acquaintance pruning become more effective under a tighter acquaintance constraint. Finally, Figure 3.3(d) manifests that SGSelect consistently outperforms SGBasic by more than one order of magnitude, even under the loosest  $k$ .

**Detailed Analysis on Proposed Strategies.** The above experiment results show that the proposed algorithm requires much less time than the baseline algorithms due to the proposed strategies. In the following, we first investigate the effectiveness of the access ordering strategy, which can guide an efficient exploration of the solution space. Figure 3.4(a) shows that either interior unfamiliarity (IU) or exterior expansibility (EE) can reduce the running time, and the proposed access ordering strategy that combines both of them leads to the greatest improvement.

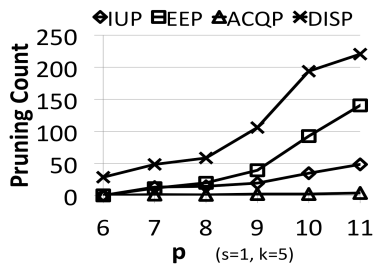
Afterwards, Figures 3.4(b), 3.4(c), and 3.4(d) analyze the pruning power of acquaint-



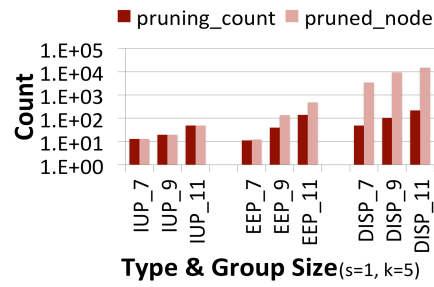
(a) Comparison of running time with different access ordering strategies used.



(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  $p$ .)

Figure 3.4: Analysis on pruning ability of proposed strategies.

tance pruning, distance pruning, interior unfamiliarity condition (IU pruning), and exterior expansibility condition (EE pruning), where a node in Figure 3.4(d) represents a visited state in the branch-and-bound tree. Note that each pruning can remove a branch in the dendrogram (i.e., remove more than one node), and the pruned node count thereby will be larger than the pruning count.

More specifically, Figure 3.4(b) first compares the running time of SGSelect with different pruning strategies. Figure 3.4(c) further analyzes the effectiveness of these pruning strategies by comparing their pruning counts in SGSelect. The distance pruning is the most effective one with 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. It is because under the same  $k$ , the number of edges required inside a size- $p$  feasible group (i.e.,  $p(p-k-1)/2$ ) increases as  $p$  grows. Therefore, the exterior expansibility condition is more difficult to hold. EE pruning thereby tends to

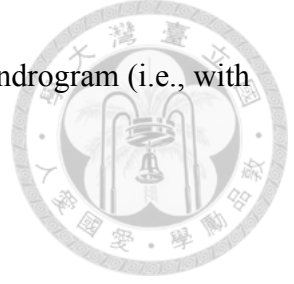


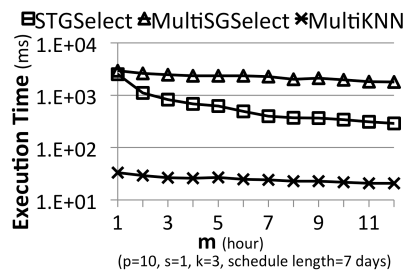
Table 3.1: The percentage of prunings located near the root of the dendrogram (i.e., with  $|V_S| \leq \lfloor \frac{p}{2} \rfloor$ ).

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

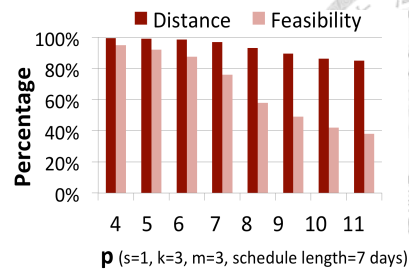
occur more frequently.

Finally, Figure 3.4(d) compares the pruned node count versus the pruning count with different strategies. The pruning ratio (i.e., the pruning count versus the pruned node count) of distance pruning reaches 1 : 90. When a pruning happens in a position closer to the root of the dendrogram, the number of pruned nodes tends to increase because those pruned nodes are all downstream nodes in the dendrogram. Therefore, we further investigate the position of pruning in different strategies. Table 3.1 shows the percentage of prunings that occur when  $|V_S| \leq \lfloor \frac{p}{2} \rfloor$ , where a smaller  $|V_S|$  implies that the pruning is closer to the root. The IU pruning usually occurs in the position more distant to the root because the pruning requires that the LHS of Eq. (3.2) exceeds 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.

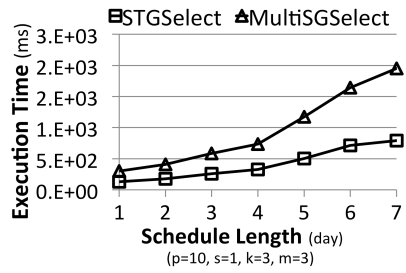
**Analysis on Temporal Dimension.** Recall that Algorithm STGSelect leverages pivot time slots to efficiently explore the temporal dimension in order to find a suitable activity time efficiently. To evaluate the performance on STGQ, we compare STGSelect with the following three algorithms: MultiSGSelect, MultiKNN, and MultiDKS, i.e., sequentially considering each candidate activity period and solving the corresponding SGQ problem using SGSelect, KNN, and DKS, respectively. Figure 3.5(a) first compares the running time of these algorithms under different activity lengths, i.e.,  $m$ . Note that the running time of MultiDKS is more than 7 hours and thereby not shown in this figure. The results show that STGSelect consistently outperforms MultiSGSelect, especially for a larger  $m$ , due to a decreasing number of pivot time slots required to be examined in STGSelect.



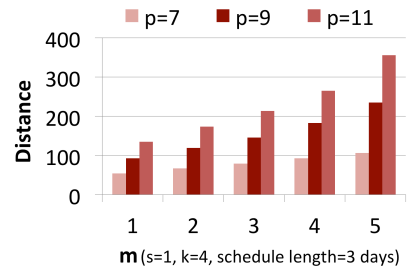
(a) Comparison of running time with different  $m$ .



(b) Solution quality analysis of KNN.



(c) Comparison of running time with different schedule lengths.



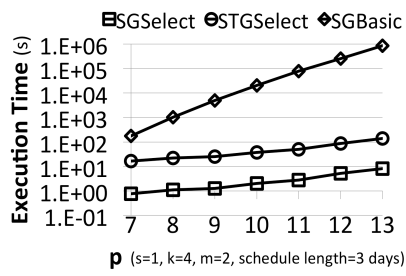
(d) Comparison of solution quality with different  $m$ .

Figure 3.5: Experimental results of STGQ.

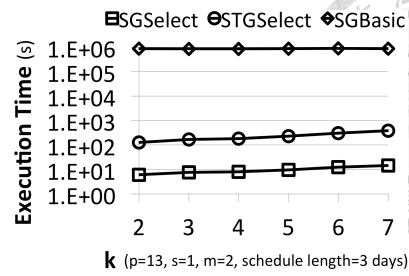
Similar to KNN, although MultiKNN is the fastest one, it is not able to guarantee the solution feasibility for the STGQ problem. Figure 3.5(b) shows that the percentage of feasible groups returned by MultiKNN drops quickly as  $p$  grows. Note that the *distance* ratio in Figure 3.5(b) remains above 85%, which is higher than 70% in Figure 3.3(b). It is because when solving STGQ, MultiKNN can only choose the candidates that are available in the activity period, rather than all the candidates in the entire social network. Therefore, the difference of the solution quality between MultiKNN and STGSelect diminishes in this case.

Figure 3.5(c) further presents the running time of STGSelect and MultiSGSelect with different lengths of schedules provided by users. More time slots need to be examined in a longer schedule. The results manifest that STGSelect consistently outperforms MultiSGSelect for varied lengths of schedules. Finally, Figure 3.5(d) analyzes the solution quality with various  $m$ . For each  $p$ , the total social distance steadily increases as  $m$  becomes larger, because a candidate vertex with a small social distance may not be available in all time slots during the examined period. It is thus necessary to choose other candidates with larger social distances.

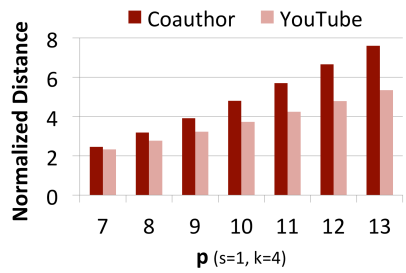




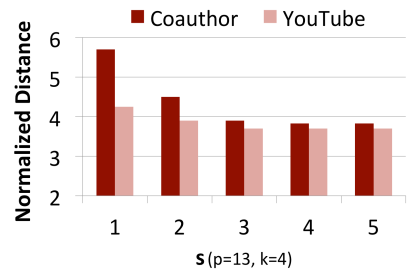
(a) Comparison of running time with different  $p$  on the YouTube dataset. (Parameters  $m$  and *schedule length* are for STGSelect.)



(b) Comparison of running time with different  $k$  on the YouTube dataset. (Parameters  $m$  and *schedule length* are for STGSelect.)



(c) Comparison of solution quality with different  $p$ .

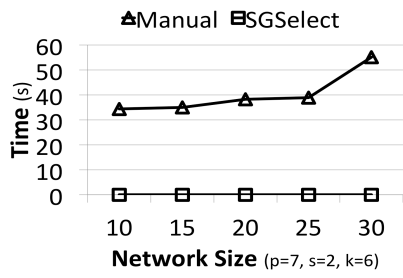


(d) Comparison of solution quality with different  $s$ .

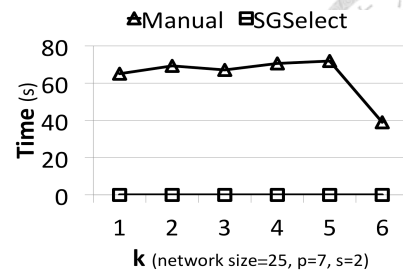
Figure 3.6: Experimental results with the YouTube dataset.

**Analysis with Large Dataset.** In the following, we compare different algorithms in a large YouTube dataset. Figure 3.6(a) manifests that the difference of running time becomes even more significant as compared to Figure 3.3(a). When  $p = 12$ , SGBasic requires more than 3 days to find the optimal solution, while SGSelect merely needs 5 seconds. When the schedules of users are considered, STGSelect finds the optimal group and a suitable activity time efficiently. In Figure 3.6(b), SGSelect still outperforms SGBasic by approximately five orders of magnitude. STGSelect, even paying extra effort to consider the user schedules, outperforms SGBasic by more than three orders of magnitude.

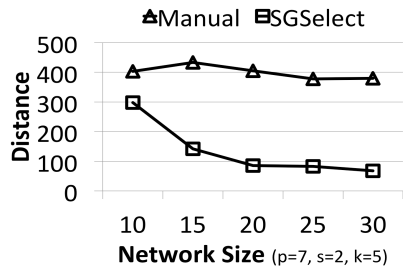
Finally, we compare the solution quality in the two different datasets in Figure 3.6(c) and Figure 3.6(d). For fair comparison, we first normalize the edge weights into the range  $[0,1]$ . Figure 3.6(c) manifests that the large YouTube dataset leads to better solution quality, and the difference becomes more significant when  $p$  increases. The reason is that more proper candidates are inclined to appear in a larger dataset, and hence there is a higher chance of forming a better group. However, as shown in Figure 3.6(d), the solution quality in the smaller Coauthor dataset can be effectively improved when the number



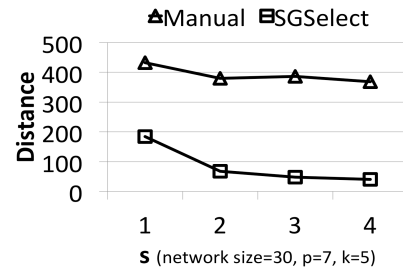
(a) Comparison of coordination time with different network sizes.



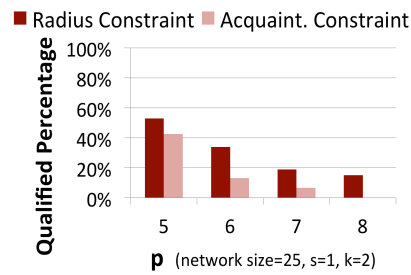
(b) Comparison of coordination time with different  $k$ .



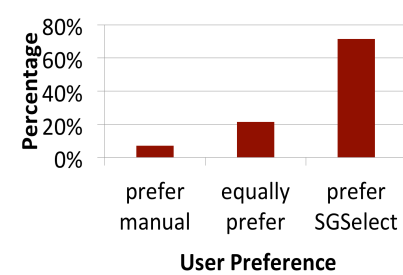
(c) Comparison of solution quality with different network sizes.



(d) Comparison of solution quality with different  $s$ .



(e) The percentage of qualified results obtained from manual coordination.



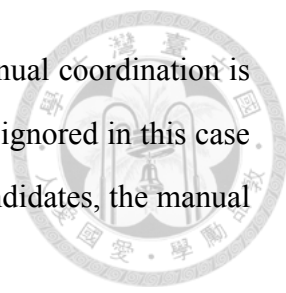
(f) The percentage of users that prefer SGSelect or manual coordination.

Figure 3.7: Experimental results of the user study.

of candidates grows as  $s$  increases.

### 3.5.3 User Study of Manual Activity Coordination

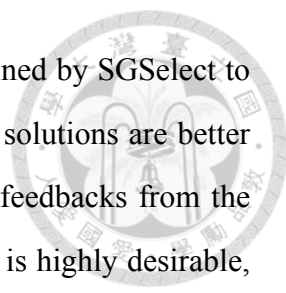
In Figures 3.7(a)-3.7(e), we compare manual coordination and SGSelect for activity planning in the user study. Figure 3.7(a) and Figure 3.7(b) first compare the coordination time with different network sizes and  $k$ , respectively. When the network size increases, the number of candidate attendees is inclined to become larger, which implies that the number of candidate groups will increase quickly. Therefore, Figure 3.7(a) manifests that the elapsed time of manual coordination grows quickly with a larger network size. Although the elapsed time of SGSelect also follows the same trend, it constantly requires less than



a few milliseconds. Figure 3.7(b) shows that the elapsed time of manual coordination is more than 60 seconds. Even when  $k = 6$ , where constraint  $k$  can be ignored in this case and the initiators only need to consider the social distances of the candidates, the manual coordination still requires approximately 40 seconds.

In Figure 3.7(c) and Figure 3.7(d), we compare the solution quality of manual coordination and SGSelect. Note that when the network size or  $s$  increases, we are able to find a group with a smaller total social distance, since more candidate attendees appear in  $V_A$ . Nevertheless, Figure 3.7(c) manifests that the improvement due to a large network size or  $s$  is tiny for manual coordination, because the activity planning problem is difficult for manual coordination even with the network size as small as 10. It is time-consuming and tedious for users to carefully extract a better group when the network size grows. In contrast, our proposed algorithm always finds the optimal one among all possible groups and outperforms the manual coordination by 67% on average. In Figure 3.7(d), the solution quality of manual coordination slightly improves when  $s$  increases from 1 to 2. However, the solution quality becomes slightly worse when  $s$  further increases from 2 to 3, which shows that the solution quality is not stable in manual coordination.

In addition to the solution quality, manual coordination suffers from the problem of solution feasibility. Given a pair of  $s$  and  $k$ , manual coordination may not always be able to find a feasible group that satisfies the social constraints even when there actually exists at least one. Figure 3.7(e) shows the percentage for manual coordination to find the answers satisfying the social constraints. Both the percentage of following the social radius constraint  $s$  and the percentage of following the acquaintance constraint  $k$  decrease quickly as  $p$  grows up, manifesting that it is much harder for the initiator to ensure the connectivity of all the attendees when  $p$  becomes larger. It is more difficult for manual coordination to ensure constraint  $k$  because examining the connectivity of all pairs of attendees is necessary to be involved, while constraint  $s$  only requires the distance calculation on each attendee and the initiator herself. In contrast, for all activity planning problems, our proposed algorithm is guaranteed to find a feasible solution (i.e., the percentage is 100%) and ensure that the total social distance is minimized.



In the user study, when we return the recommended groups obtained by SGSelect to the users, Figure 3.7(f) manifests that 93% of users agree that these solutions are better or as good as the groups derived by themselves. According to the feedbacks from the users, we conclude that an automatic group recommendation system is highly desirable, since it reduces considerable efforts on activity planning. This has a significant impact on increasing users' willingness to organize social activities with friends.

### 3.6 Summary

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. 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. Experimental results indicate that the proposed SGSelect and STGSelect are significantly more efficient and scalable than the baseline approaches. According to a user study we performed, the proposed algorithm obtains higher solution quality with much less coordination effort compared with manual activity coordination, and hence increases users' willingness to organize activities with friends.

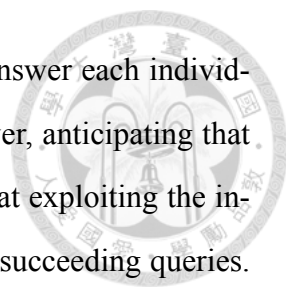


## Chapter 4

# Efficient Processing of Consecutive Group Queries for Social Activity Planning

### 4.1 Introduction

In Chapter 3, we have introduced the Social Group Query (SGQ) and Social-Temporal Group Query (STGQ) for activity planning. Note that it is difficult for a user to specify all the query parameters right at once 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 consecutive 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 [2, 6, 18, 24, 27, 41, 43, 55, 59]. Some existing studies [14, 34, 63, 64] 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 succeeding queries.



A straightforward method to support a sequence of SGQs is to answer each individual query with Algorithm SGSelect introduced in Chapter 3. 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 succeeding queries. To facilitate the above idea, in this chapter, we propose *Consecutive Social Group Query (CSGQ)*, which aims to efficiently support a sequence of SGQs with varying parameters, and CSGQ can be extended to support a sequence of STGQs. Accordingly we design a new tree 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 a new index structure, called *Social Boundary*, which effectively indexes the intermediate solutions required for processing each CSGQ 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 CSGQ only processes a small portion of candidate subgroups in a social boundary.

Contributions of this chapter are summarized as follows.

- We observe the needs of activity initiators who are inclined to adjust the social constraints for alternative recommendations in order to finalize the invitation. Thus, we propose a new query, namely, CSGQ, to support a sequence of social group queries with a variety of query parameters. We design two new data structures, *accumulative search tree* and *social boundary*, to accelerate a sequence of group queries. The proposed data structures effectively cache and index intermediate solutions in preceding queries to support succeeding ones. We also extend CSGQ to consider the temporal dimension.
- We derive a set of node selection rules to efficiently and accurately build the social boundaries under various query parameters. We further prove that the derived node selection rules obtain a sufficient and necessary set of nodes for each social boundary. We also perform a series of sensitivity tests to evaluate the performance

of proposed mechanisms for accelerating CSGQ. Experimental results show that the caching mechanisms are able to substantially reduce the query processing time.

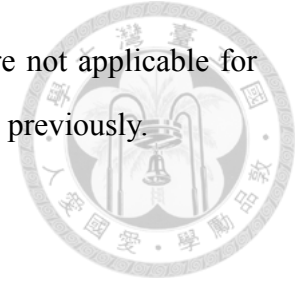
The rest of this chapter is summarized as follows. In Section 4.2, we introduce related works. After that, we introduce CSGQ and the caching mechanisms used to support consecutive queries in Section 4.3. We then detail the construction and maintenance of the caching mechanisms in Section 4.4. Proof of solution optimality and extensions of CSGQ are provided in Section 4.5. Finally, we present the experimental results in Section 4.6 and summarize this chapter in Section 4.7.

## 4.2 Related Works

In the field of social network analysis, there are many studies related to SGQ. For example, research on finding various kinds of subgroups, such as clique, k-plex and k-truss has been conducted (e.g., [6, 18, 43, 55, 59]). There are also related works on group formation (e.g., [3, 44, 57]), team formation (e.g., [2, 24, 41]), and group query (e.g., [27, 28, 60]). Moreover, there are some related works on community search and social circle discovering (e.g., [35, 52, 61]). However, none of them explore consecutive queries, which are crucial in real scenarios.

For diversified query, some existing studies (e.g., [14, 34, 63, 64]) return multiple subgraphs with diverse characteristics in one single query. However, since these studies are not specifically designed for activity planning, social connectivity and tightness are not their major concern. Therefore, the returned subgroups are not guaranteed to achieve social cohesiveness. Moreover, without feedback and guidance from user-specified parameters, most returned subgraphs in the diversified query are likely to be redundant (i.e., distant from the desired results of users). In addition, none of the aforementioned studies consider the temporal dimension to facilitate automatic activity planning. On the other hand, session query and reinforcement learning in retrieval (e.g., [20, 26, 50]) that allow users to tailor the query have attracted increasing attentions. However, these studies are designed for document retrieval and hence cannot handle the social network graph and

user schedules. Therefore, these aforementioned research works are not applicable for automatic activity planning, and the CSGQ problem is not addressed previously.



### 4.3 Consecutive 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 Section 3.5.3), 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 [2, 6, 18, 24, 27, 41, 43, 55, 59]. A naive method to process a sequence of 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 consecutive SGQs by caching intermediate solutions for reuse. Therefore, we propose two mechanisms, *Accumulative Search Tree (AST)* and *Social Boundary (SB)*, to support processing of *Consecutive Social Group Queries (CSGQs)*. 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 the succeeding 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 succeeding SGQs to start with intermediate solutions. In the following, Section 4.3.1 first introduces AST and SB. Section 4.3.2 then presents how to exploit AST and SB to answer CSGQs efficiently. The construction and maintenance of AST and SB will be detailed later in Section 4.4. After that, we will prove the solution optimality and extend CSGQ to support the temporal dimension in Section 4.5.



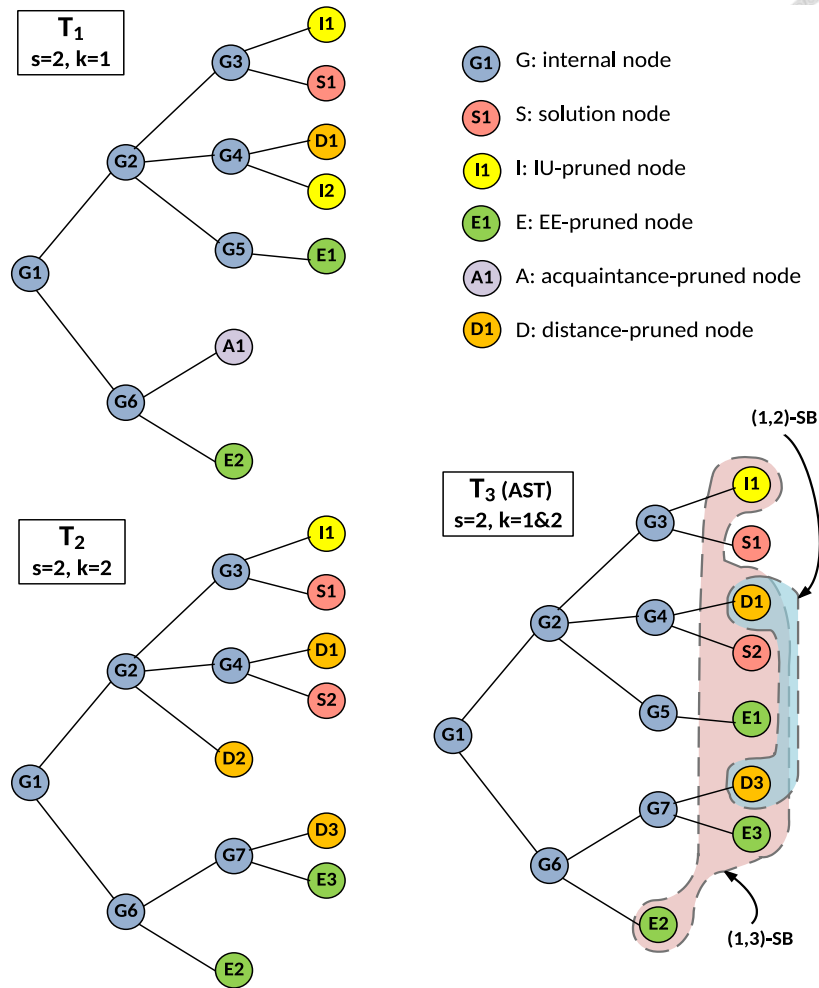


Figure 4.1:  $T_1$  and  $T_2$  are two dendrograms with different  $k$ , and  $T_3$  is the accumulative search tree.

### 4.3.1 Accumulative Search Tree and Social Boundary

Figure 4.1 illustrates two search trees  $T_1$  and  $T_2$  corresponding to two SGQs with slightly different parameters  $(s, k) = (2, 1)$  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 consecutively in the future. Also, to support quick traversal of the tree in the consecutive 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 4.3.1.** *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 Figure 4.1 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  $G_i$  and  $S_i$  stand for an internal node and a solution node, respectively. When processing the succeeding 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 the consecutive queries, and we propose SB to address this issue.

**Definition 4.3.2.** *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.<sup>1</sup> 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 Figure 4.1, 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 4.1 with the query example in Figure 4.2. The content of this table is filled using the nodes in AST after the first SGQ is processed. For each succeeding 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 computational 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 computational 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 4.3.2. We will then detail the construction of SBs in Section 4.4 and prove

<sup>1</sup>The range of possible  $s$  and  $k$  are  $1 \leq s \leq s_{max}$  and  $0 \leq k \leq p - 1$ , 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 4.1: The  $(s,k)$ -SBs constructed after the first SGQ.

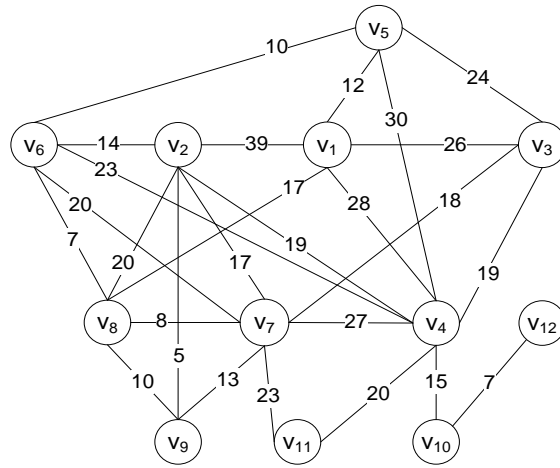
$(s,k)$	Nodes indexed by the $(s,k)$ -SB
...	...
(2,0)	P23, P27, P28, P29
(2,1)	P20, P23, P27, P28, P29 [S1]
(2,2)	P4, P11, P16, P18, P20, P22, P23, P27, P28, P29 [S1]
(2,3)	P2, P4, P9, P11, P16, P18, P20, P22, P23, P27, P28, P29 [S2]
(2,4)	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, 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 Theorem 4.5.1 that the nodes indexed by  $(s,k)$ -SB are sufficient for finding the optimal solution.

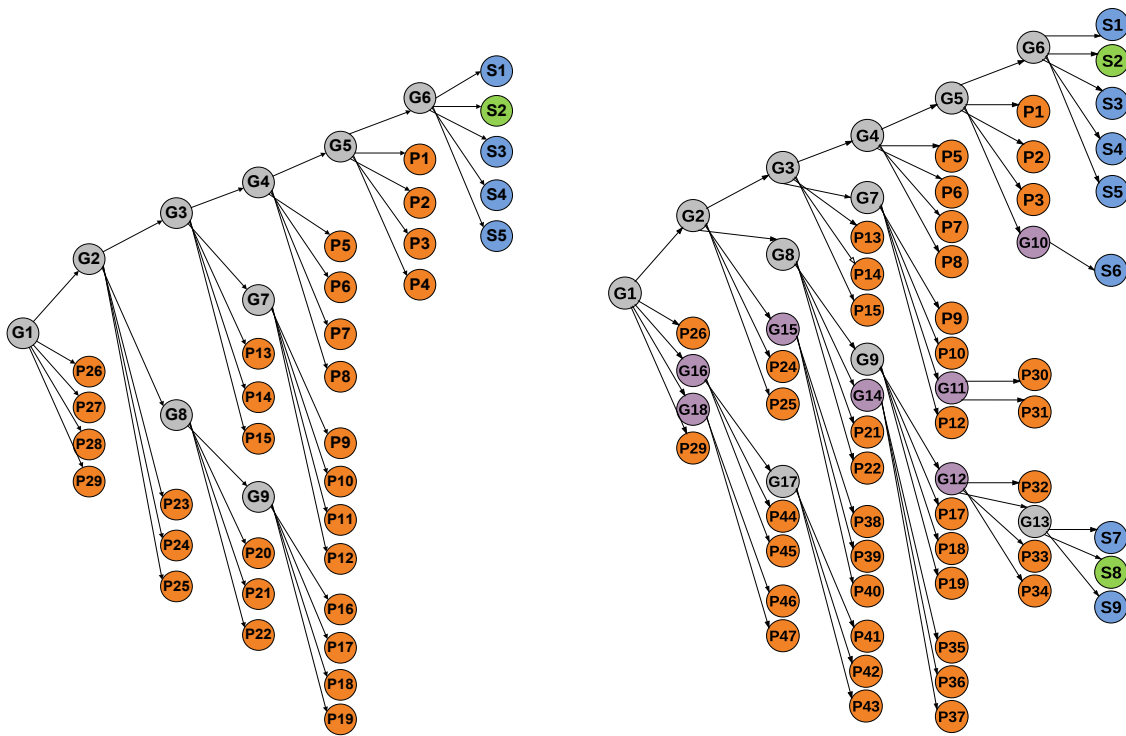
### 4.3.2 Solution Acquisition Using AST and SB

For each new SGQ with a specified pair of  $s$  and  $k$ , we first use the  $(s,k)$ -SB to quickly identify the corresponding nodes 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 B): 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 succeeding SGQ, 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 4.4.

**Example 4.3.1.** In this example,  $v_7$  in Figure 4.2(a) is the initiator. After processing the first query with  $(p, s, k) = (7, 3, 3)$  using SGSelect, the initial AST is shown in Figure



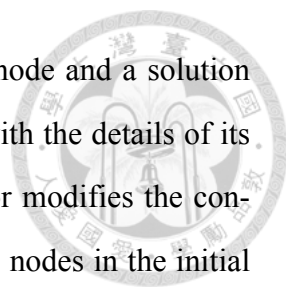
(a)



(b)

(c)

Figure 4.2: An illustrative example for CSGQ. (a) The sample social network, (b) the initial accumulative search tree and (c) the accumulative search tree after the second query.



4.2(b). Nodes  $G_i$ ,  $P_i$  and  $S_i$  stand for an internal node, a pruned node and a solution node, respectively. In Table 4.1, we list SBs for different  $s$  and  $k$ , with the details of its construction to be provided in Section 4.4. 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  $S_1$ . 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  $P_1$ . 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  $S_1$  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 Figure 4.2(c). For example, a distance-pruned node  $P_4$  in Figure 4.2(b) is successfully expanded into the internal node  $G_{10}$  in Figure 4.2(c), which eventually is expanded into a new solution node  $S_6$ . After processing all the 10 pruned nodes in (3,2)-SB, we obtain four new feasible solutions (i.e.,  $S_6$ ,  $S_7$ ,  $S_8$  and  $S_9$ ). Among them and the existing solution (i.e.,  $S_1$ ),  $S_8$  is returned as the optimal one since it has the smallest total social distance. ■

## 4.4 Index Construction and Maintenance

Section 4.3 has illustrated how to exploit AST and SB to answer CSGQs. In the following, we further detail how to construct a table of  $(s,k)$ -SBs for various  $s$  and  $k$  in Section 4.4.1. The updating procedure of AST and SB is then discussed in Section 4.4.2.

### 4.4.1 Node Indexing of AST Using SB

To effectively reduce redundant node processing in CSGQs, 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 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  $k$  in Rule 1 and then return to the social radius constraint  $s$  in Rule 2.

**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. (3.2), (3.3), (3.4) and (3.5) in Section 3.3.2, 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

$$k' \geq \max\{U(V_S), k + 1\}. \quad (4.1)$$

**Example 4.4.1.** Figure 4.2(b) 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. (4.1), 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. (3.5) 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 succeeding SGQs.

- *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 constraint.



ilarity 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 3.3.2, 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.<sup>2</sup> We then update the social distance according to different  $s'$  to

<sup>2</sup>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  $d_{v,q}^i < \infty$ .

keep track of the status of the pruned node.<sup>3</sup> An  $(s',k)$ -SB includes an EE-pruned node only if its  $V_A^{s'}$  is large enough such that  $A(V_S) \geq p - |V_S|$ , implying that Eq. (3.3) does not hold and prevents EE pruning.

**Example 4.4.2.** Figure 4.2(b) 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. (3.3) 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 the succeeding 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. Example 4.4.3 illustrates how the base node and delta nodes work.

<sup>3</sup>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  $d_{v,q}^i$  with  $i = s'$ .



Table 4.2: (a) The social distances from different vertices to  $v_7$  under various  $s$ , and (b) variations of node  $P_5$ .

	$v_1$	$v_4$	$v_5$	$v_6$	$v_9$	$v_{10}$	$v_{11}$	$v_{12}$
(a) $s = 1$	$\infty$	27	$\infty$	20	13	$\infty$	23	$\infty$
$s = 2$	25	27	30	15	13	42	23	$\infty$
$s = 3$	25	27	25	15	13	42	23	49
$s = 4$	25	27	25	15	13	42	23	49

(b) $P_5$ : $\langle V_S = \{v_6(20), v_7(0)\}, V_A = \{v_4(27), v_9(13), v_{11}(23)\} \rangle$
$C_{P_5}^2$ : $\langle V_S^2 = \{v_6(15), v_7(0)\}, V_A^2 = \{v_1(25), v_4(27), v_5(30), v_9(13), v_{10}(42), v_{11}(23)\} \rangle$
$C_{P_5}^3$ : $\langle V_S^3 = \{v_6(15), v_7(0)\}, V_A^3 = \{v_1(25), v_4(27), v_5(25), v_9(13), v_{10}(42), v_{11}(23), v_{12}(49)\} \rangle$
$C_{P_5}^4$ : $\langle V_S^4 = \{v_6(15), v_7(0)\}, V_A^4 = \{v_1(25), v_4(27), v_5(25), v_9(13), v_{10}(42), v_{11}(23), v_{12}(49)\} \rangle$
$B_{P_5}^{2-4}$ : $\langle V_S^2 = \{v_6(15), v_7(0)\}, V_A^2 = \{v_1(25), v_4(27), v_5(30), v_9(13), v_{10}(42), v_{11}(23)\} \rangle$
$D_{P_5}^{3-4}$ : $\langle V_S' = \emptyset, V_A' = \{v_5(-5), v_{12}(49)\} \rangle$

**Example 4.4.3.** This example employs a query with a smaller group size  $p$  to illustrate the function of the base node and delta nodes. Assume that  $v_7$  in Figure 4.2(a) issues a query with  $(p, s, k) = (4, 1, 1)$ . With the radius graph extraction, we obtain the social distance of each vertex under different  $s$ . Part of the results are listed in Table 4.2(a), and the social distance of a vertex may decrease as  $s$  increases. Moreover, a vertex is included as a candidate in  $V_A^{s'}$  if its social distance becomes smaller than  $\infty$  under the social constraint  $s = s'$ . Here we consider an EE-pruned node  $P_5$  with  $V_S = \{v_6(20), v_7(0)\}$  and  $V_A = \{v_4(27), v_9(13), v_{11}(23)\}$  generated in the query as an example. (The number in the parentheses next to  $v_i$  is the social distance from  $v_i$  to the initiator  $v_7$ .) A naive approach to handle various  $s'$  is generating standalone copies of  $P_5$  for each  $s'$  from  $s + 1 = 2$  to  $s_{max} = 4$ , i.e.,  $C_{P_5}^2$ ,  $C_{P_5}^3$  and  $C_{P_5}^4$  in Table 4.2(b), where  $s_{max}$  is the largest possible  $s$ . However, we observe that there is a large overlap among  $C_{P_5}^2$ ,  $C_{P_5}^3$  and  $C_{P_5}^4$ . Therefore, in the following, we will show how to condensedly maintain these copies using the base node and the delta nodes.

First, the base node  $B_{P_5}^{2-4}$  contains the  $V_S^{s'}$  and  $V_A^{s'}$  of  $P_5$  with the smallest  $s'$  (i.e., 2). Here the index 2 – 4 means this node is used when reconstructing the  $V_S^{s'}$  and  $V_A^{s'}$  with  $s' = 2, 3$ , or 4. When  $s'$  increases to 3, it is necessary to record the newly joined vertex (i.e.,  $v_{12}$ ) and the difference of social distance (i.e.,  $-5$  for  $v_5$ ) in the

delta node  $D_{P_5}^{3-4}$ . The unchanged vertices can be omitted to save space. Since there is no further change when  $s' = 4$ , more delta nodes do not need to be generated. When a new query comes in, we only need to take the base node and use delta nodes to add new candidates or update the social distance as necessary. Thus, the  $V_S^{s'}$  and  $V_A^{s'}$  that fit the new social radius constraint can be dynamically generated when needed. ■

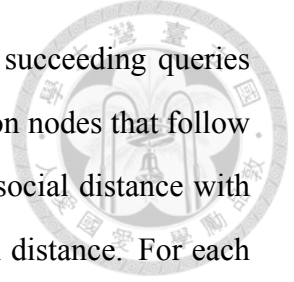
- *Acquaintance-pruned nodes.* Similar to the EE-pruned nodes, some acquaintance-pruned nodes may be expanded into new sub-trees when  $s' > s$ , since new candidate attendees may appear in  $V_A$ . Specifically, an  $(s', k)$ -SB includes an acquaintance-pruned node only if its  $V_A^{s'}$  is large 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. (3.5) 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 succeeding 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.

**Rule 3: node indexing for different  $s$  and  $k$**  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. Therefore, we explore the generalized case as follows.

**(1) Pruned nodes.**

- *IU-pruned nodes.* Rule 1-(1) indicates that an IU-pruned node is included in the  $(s,k')$ -SB if  $k' \geq U(V_S)$ .<sup>4</sup> Rule 2-(1) shows that, when  $k$  is fixed, an IU-pruned node is not in any  $(s',k)$ -SB regardless of  $s'$ ; however, when constructing  $(s',k')$ -SBs,  $s'$  can still to reduce the number of IU-pruned nodes. That is, a pruned node is included in an  $(s',k')$ -SB only if all vertices in its  $V_S$  are within  $s'$  hops of the initiator, i.e.,  $\max_{v \in V_S} h_v \leq s'$ . Combining the inequalities, an IU-pruned node is included in an  $(s',k')$ -SB only if  $k' \geq U(V_S)$  and  $\max_{v \in V_S} h_v \leq s'$ .

**Example 4.4.4.** We revisit  $P1$  in Example 4.4.1, with  $V_S = \{v_2, v_6, v_7, v_8, v_9, v_{11}\}$

<sup>4</sup>The original rule listed in Rule 1-(1) is  $k' \geq \max\{U(V_S), k + 1\}$ . However, it can be simplified by observing that  $U(V_S)$  must exceed  $k$ ; otherwise, the IU pruning will not happen. Similarly, the rules of EE-pruned and acquaintance-pruned nodes used later are also simplified.

and  $V_A = \{v_1, v_4, v_5, v_{10}\}$ . Since  $U(V_S) = 4$  was already derived in the first query,  $P1$  will only appear in  $(s', k')$ -SBs with  $k' \geq 4$ . Moreover, because the vertices in  $V_S$  are all within one hop of the initiator,  $\max_{v \in V_S} h_v = 1$  holds. Therefore, the IU-pruned node  $P1$  will only appear in  $(s', k')$ -SBs with  $1 \leq s'$  and  $k' \geq 4$ , such as (2,4)-SB and (3,4)-SB in Table 4.1. ■

- *EE-pruned nodes.* According to Rule 1-(1), an EE-pruned node is included in the  $(s, k')$ -SB if

$$k' \geq p - |V_S| - A(V_S) + k. \quad (4.2)$$

We employ the social radius constraint to reduce the number of nodes included. Specifically, it is not necessary to keep the pruned nodes for the  $(s', k')$ -SBs with  $s' < \max_{v \in V_S} h_v$ , even if their  $k'$  satisfies Eq. (4.2). Other  $(s', k')$ -SBs whose  $k'$  does not satisfy Eq. (4.2) can include an EE-pruned node only if their  $s'$  is large enough so that its  $A(V_S)$  with new candidates in  $V_A^{s'}$  is no smaller than  $p - |V_S|$ , implying that this node will not be pruned by EE pruning again.

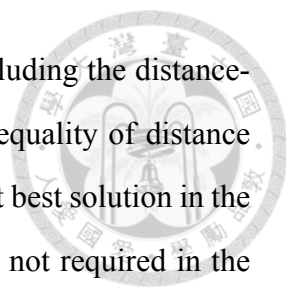
- *Acquaintance-pruned nodes.* According to Rule 1-(1), an acquaintance-pruned node is included in the  $(s, k')$ -SB if

$$k' \geq p - |V_S| - 1 - \sum_{v \in M_A} |V_A \cap N_v| / (p - |V_S|). \quad (4.3)$$

We again use the social radius constraint to reduce the number of nodes included. The pruned nodes for the  $(s', k')$ -SBs with  $s' < \max_{v \in V_S} h_v$  are not needed, even if their  $k'$  satisfies Eq. (4.3). Other  $(s', k')$ -SBs whose  $k'$  does not satisfy Eq. (4.3) can include an acquaintance-pruned node only if their  $s'$  is large enough such that

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

- *Distance-pruned nodes.* A distance-pruned node may be successfully expanded when  $k' < k$  or when  $s' \neq s$ , according to Rule 1-(1) or Rule 2-(1), respectively.



Similarly, we can reduce the number of included nodes by excluding the distance-pruned nodes with  $\max_{v \in V_S} h_v \leq s'$ . We further reuse the inequality of distance pruning strategy (i.e., Eq. (3.4)) by replacing  $D$  with the current best solution in the  $(s', k')$ -SB. If the inequality holds, the distance-pruned node is not required in the SB since it will be pruned again.

**(2) Solution nodes.** Combining Rule 1-(2) and Rule 2-(2), a solution node is included in the  $(s', k')$ -SBs with  $k' \geq U(V_S)$  and  $s' \geq \max_{v \in V_S} h_v$ , i.e., satisfying the acquaintance and social radius constraints, respectively.

**(3) Internal nodes.** Rule 2-(3) indicates that the  $(s', k)$ -SBs with  $s' > s$  should include internal nodes for new candidates due to the increment of  $s'$ . For changes in  $k'$ , the internal nodes with  $U(V_S) > k'$  violate the acquaintance constraint  $k'$  and does not generate a solution. Therefore, the  $(s', k')$ -SB only includes the internal nodes if  $k' \geq U(V_S)$  and  $s' \geq \max_{v \in V_S} h_v$ .

#### 4.4.2 Updating of AST and SB

After the first SGQ is processed, the initial AST is generated, and the  $(s, k)$ -SBs for various  $s$  and  $k$  are constructed using the node selection rules derived in Section 4.4 to facilitate the processing of the succeeding SGQs. In the following, we detail the update of AST and SBs while processing a succeeding SGQ. During the process, some nodes are expanded into new sub-trees in the AST, and these sub-trees may include new solution, internal and pruned nodes. All these newly generated nodes are added into the existing AST to retain the latest intermediate solutions. Since a tree node may reside in more than one SB, the successfully expanded nodes may still reside in some other  $(s, k)$ -SBs not yet processed to solve any query. To keep the latest intermediate solutions, certain new nodes are selected to be included in some unprocessed  $(s, k)$ -SBs based on the node selection rules derived in Section 4.4, which can effectively identify a small portion of nodes that need to be processed in the query with corresponding  $s$  and  $k$ .

While updating, we can save the space by considering the  $s$  of all historical queries



Table 4.3: The  $(s,k)$ -SBs after updated according to the second SGQ.

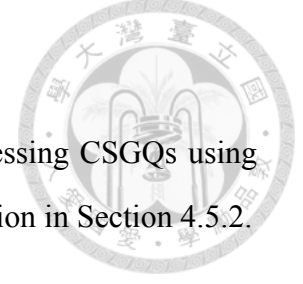
$(s,k)$	Nodes indexed by the $(s,k)$ -SB
...	...
(3,0)	P29, P44
(3,1)	P29, P41, P44 [S1]
(3,2)	[already processed]
(3,3)	[already processed]
(3,4)	P1, P2, P3, P5, P6, P7, P8, P9, P10, P12, P13, P14, P15, P17, P18, P19, P21, P22, P24, P25, P26, P29, P30, P31, P32, P33, P34, P35, P36, P37, P38, P39, P40, P41, P42, P43, P44, P45, P46, P47 [S8]
(3,5)	...
(3,6)	...
...	...

simultaneously. Consider an example with two historical queries, where  $s$  of the first is 3, and  $s$  of the second is 1. When updating the SBs, if we only compare  $s'$  of the SBs with  $s$  of the latest query (i.e., 1), it is necessary to include the new internal nodes in the  $(s',k')$ -SB with  $s' > 1$ . However, it is redundant to include these nodes in the  $(s',k')$ -SBs with  $1 < s' \leq 3$ , since the nodes containing the candidates within 3 hops of the initiator were generated in the first query and already exist in the AST and SBs. Therefore, when we decide if the internal nodes should be included in an  $(s',k')$ -SB, it is important to consider the  $s$  of all historical queries, not just the latest. To be specific, let  $s_{old\_max}$  denote the largest  $s$  among all historical queries, only the  $(s',k')$ -SBs with  $s' > s_{old\_max}$  are required to consider if the new internal nodes need to be included. In the following, we provide an example to illustrate the update of SBs.

**Example 4.4.5.** In Example 4.3.1, after processing the succeeding SGQ, any successfully expanded node is replaced by the branches generated (as shown in Figure 4.2(c)) such that these new results can be leveraged in processing future queries. When AST changes, SBs in Table 4.1 should be updated accordingly. For example, pruned nodes  $P23$ ,  $P27$  and  $P28$  are removed from the SBs since they were expanded, while a new pruned node  $P44$  is now included in (3,1)-SB according to the node selection rules. The updated SBs after processing the succeeding SGQ with  $(s, k) = (3, 2)$  are listed in Table 4.3. ■



## 4.5 Solution Optimality and Extensions



In Section 4.5.1, we first prove the solution optimality of processing CSGQs using AST and SB. We then extend CSGQ to support the temporal dimension in Section 4.5.2.

### 4.5.1 Solution Optimality

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 4.5.1.** *Processing the nodes in the  $(s', k')$ -SB obtains the optimal solution to the CSGQ.*

*Proof.* Here we provide a sketch of proof, and the complete version can be found in Appendix A.1. Let  $SP(s)$  denote the solution space consisting of all size- $p$  feasible solutions within  $s$  hops. AST covers  $SP(s_{old\_max})$ , where  $s_{old\_max}$  is the largest  $s$  among all processed historical queries, because feasible solutions are either the existing solution nodes or descendant nodes of unexpanded pruned nodes in AST. If we expand the cached pruned nodes to generate the solutions and compare them with the existing ones, the optimal solution of the query can be obtained.

However, an  $(s', k')$ -SB only includes a small portion of cached nodes chosen by the node selection rules. In the following, we prove that all the pruned nodes not included in an  $(s', k')$ -SB do not generate the optimal solution to the query with specified  $s'$  and  $k'$ . Specifically, an EE-pruned node does not stay in an  $(s', k')$ -SB if  $k'$  is not large enough such that

$$k' < p - |V_S| - A(V_S) + k, \quad (4.4)$$

and  $s'$  is also not large enough such that

$$A(V_S) < p - |V_S|. \quad (4.5)$$



Combining Eq. (3.1) and Eq. (4.4), we have

$$k' < p - |V_S| - \min_{v \in V_S} \{|V_A \cap N_v| + (k - |V_S - v - N_v|)\} + k,$$

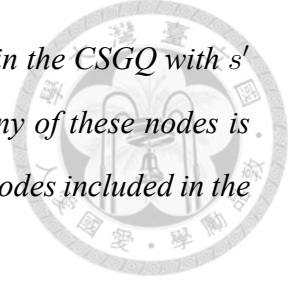
or equivalently,

$$\min_{v \in V_S} \{|V_A \cap N_v| + (k' - |V_S - v - N_v|)\} < p - |V_S|,$$

which indicates that  $k'$  is not loose enough for this node to satisfy the exterior expansibility condition, and hence this node must be pruned again. Eq. (4.5) indicates that  $s'$  is too small to bring a sufficient number of new vertices, which also implies that the exterior expansibility condition is not satisfied, and hence this node must be trimmed off again. Similarly, the acquaintance and distance pruned nodes that are not chosen by the node selection rules cannot be successfully expanded for better solution, as derived in the complete proof (see Appendix A.1).

Moreover, a solution node does not reside in an  $(s', k')$ -SB only if it violates the social constraints or if there already exists another solution node with a smaller total social distance in the SB. Therefore, these nodes can be omitted without missing solution optimality. When  $s' > s_{old\_max}$ , new feasible solutions are necessary to be examined because  $SP(s') \supseteq SP(s_{old\_max})$ . The  $(s', k')$ -SBs with  $s' > s_{old\_max}$  include the internal nodes in AST, and the  $V_S$  in these internal nodes are expanded with newly added candidates to generate new feasible solutions. Therefore, the solutions in  $SP(s') - SP(s_{old\_max})$  are also carefully examined. The theorem follows.  $\square$

In the following, we further prove that all the nodes included in the SBs are feasible for further expansion in Theorem 4.5.2, and then we show that the nodes included in the SBs are sufficient and necessary in Corollary 4.5.1. Since all the infeasible nodes are excluded from the SBs, the nodes to be processed in the succeeding queries can be minimized. This property brings a considerable improvement in the efficiency of query processing, as shown later in Section 4.6.1.



**Theorem 4.5.2.** *The nodes included in the  $(s',k')$ -SB are expansible in the CSGQ with  $s'$  and  $k'$ , i.e., they can be expanded to investigate new solutions. If any of these nodes is discarded, the optimal solution is no longer guaranteed. That is, the nodes included in the  $(s',k')$ -SB are necessary.*

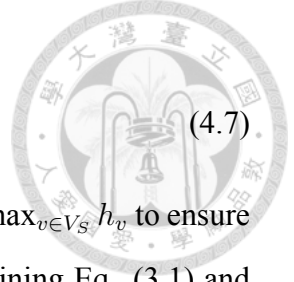
*Proof.* (1) An internal node is included in the  $(s',k')$ -SB only when  $s' > s_{old\_max}$  and  $k' \geq U(V_S)$ , where  $s_{old\_max}$  is the largest  $s$  among all processed historical queries. Note that all the vertices in the  $V_S$  of this internal node are within  $s$  hops from the initiator. Meanwhile, they must also be located within  $s_{old\_max}$  hops from the initiator since  $s_{old\_max} \geq s$  for any internal node. If  $s' > s_{old\_max}$ , the internal node that satisfies the social radius constraint  $s_{old\_max}$  must also satisfy the loosened new social radius constraint  $s'$ . Furthermore, the inequality  $k' \geq U(V_S)$  implies that this internal node also satisfies the acquaintance constraint. Therefore, this internal node is expansible under the specified  $s'$  and  $k'$  of the new query. In other words, new feasible solutions and even the optimal solution could be generated by adding newly joined candidates. If this internal node is not kept in the  $(s',k')$ -SB, the optimal solution is not guaranteed.

(2) A solution node is included in the  $(s',k')$ -SB when  $k' \geq U(V_S)$  and  $s' \geq \max_{v \in V_S} h_v$ , in order to ensure that this solution node satisfies the acquaintance constraint and the social radius constraint, respectively. Moreover, the solution node maintained in the  $(s',k')$ -SB is the optimal solution obtained so far. Therefore, it is essential to take into account this solution node in order to guarantee the optimal solution.

(3) The same as the solution nodes, an IU-pruned node will be included in the  $(s',k')$ -SBs when  $k' \geq U(V_S)$  and  $s' \geq \max_{v \in V_S} h_v$ , in order to ensure that this IU-pruned node satisfies the acquaintance constraint and the social radius constraint, respectively. Therefore, this IU-pruned node is expansible under the social constraints, and new feasible solutions could be obtained by expanding this node. If this node is not included in the  $(s',k')$ -SB, the optimal solution is not guaranteed to be found.

An EE-pruned node can be included in the  $(s',k')$ -SBs if  $k'$  is large enough such that

$$k' \geq p - |V_S| - A(V_S) + k, \quad (4.6)$$



or  $s'$  is large enough such that

$$A(V_S) \geq p - |V_S|. \quad (4.7)$$

In addition to Eq. (4.6) or Eq. (4.7), the  $(s', k')$ -SBs also satisfy  $s' \geq \max_{v \in V_S} h_v$  to ensure the EE-pruned node satisfying the social radius constraint  $s'$ . Combining Eq. (3.1) and Eq. (4.6), we have

$$k' \geq p - |V_S| - \min_{v \in V_S} \{|V_A \cap N_v| + (k - |V_S - v - N_v|)\} + k,$$

or equivalently,

$$\min_{v \in V_S} \{|V_A \cap N_v| + (k' - |V_S - v - N_v|)\} \geq p - |V_S|,$$

which means that  $k'$  is loose enough, and this node does not need to be pruned by the EE pruning again. Eq. (4.7) also indicates that  $s'$  is large enough such that this node is not necessary to be pruned by the EE pruning again, and new feasible solutions could be generated by expanding this node. Therefore, if this expansible EE-pruned node is not included, the optimal solution is not guaranteed to be found.

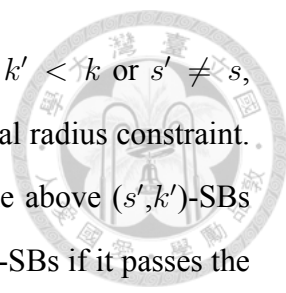
Similarly, an acquaintance-pruned node is included in the  $(s', k')$ -SBs if  $k'$  is large enough such that

$$\sum_{v \in M_A^{s'}} |V_A^{s'} \cap N_v| \geq (p - |V_S|)(p - |V_S| - k' - 1), \quad (4.8)$$

and  $s'$  is large enough such that

$$s' \geq \max_{v \in V_S} h_v. \quad (4.9)$$

Eq. (4.8) indicates that this node is not pruned by the acquaintance pruning under the loosened acquaintance constraint  $k'$ , while Eq. (4.9) ensures that this node satisfies the social radius constraint. Therefore, this node is expansible in the new query and could lead to feasible solutions. If this expansible acquaintance-pruned node is not included, the optimal solution is not guaranteed to be found.



A distance-pruned node can be included in the  $(s',k')$ -SBs with  $k' < k$  or  $s' \neq s$ , together with  $s' \geq \max_{v \in V_S} h_v$  to ensure this node satisfying the social radius constraint. This pruned node is then compared with the existing solution on the above  $(s',k')$ -SBs with the distance pruning strategy. The node is included in the  $(s',k')$ -SBs if it passes the distance pruning, i.e., it may be expanded into new feasible solutions with smaller total social distances than the existing solutions in these  $(s',k')$ -SBs. Therefore, if this distance-pruned node is not included, the optimal solution is not guaranteed to be found.

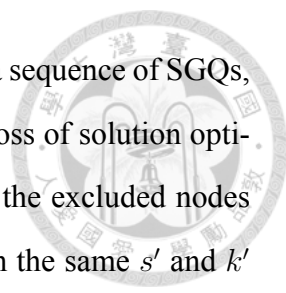
Note that in addition to the type of pruning with which a node is originally removed, we also examine this pruned node under the other three types of pruning strategies with specified  $s'$  and  $k'$  of the new query. Therefore, this pruned node is ensured to be expansible and could generate feasible solutions of the new query, and the optimal solution may be missed if this pruned node is not included in the  $(s',k')$ -SB. The theorem follows.  $\square$

**Corollary 4.5.1.** *The nodes included in the  $(s',k')$ -SB are sufficient and necessary for obtaining the optimal solution of the CSGQ with specified  $s'$  and  $k'$ .*

*Proof.* Theorem 4.5.1 shows that processing the nodes in the  $(s',k')$ -SB obtains the optimal solution to the CSGQ with specified  $s'$  and  $k'$ , which implies that the nodes included in the  $(s',k')$ -SB are sufficient. Theorem 4.5.2 then shows that the nodes included in the  $(s',k')$ -SB are necessary. The corollary follows.  $\square$

## 4.5.2 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 3.4, 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 4.4 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 4.5.1, all the excluded nodes still cannot generate the optimal solution to a succeeding STGQ with the same  $s'$  and  $k'$  as with the succeeding 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.

## 4.6 Experimental Results

In this section, we analyze the experimental results of CSGQ with the proposed caching mechanisms (i.e., AST and SB) using real datasets. We first perform a series of sensitivity tests to study the impact of query parameters with a coauthorship network (called Coauthor) of 16,726 people [38]. In addition to the sensitivity tests, we also conduct experiments on a much larger YouTube social network [61], which includes 1,134,890 people. To evaluate CSGQ thoroughly, we conduct experiments with various parameter settings for CSGQ. Specifically, the parameters are sequentially increased or decreased with differ-

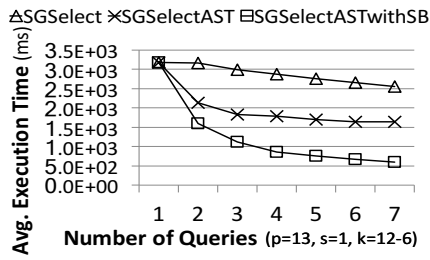
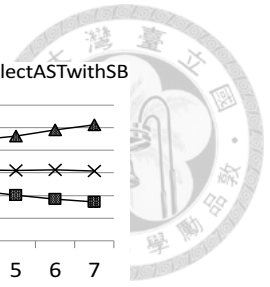
ent rates. Moreover, we also conduct experiments with randomly parameter adjustment. The algorithms are implemented on an HP DL580 server with four Intel E7-4870 2.4 GHz CPUs and 128 GB RAM.



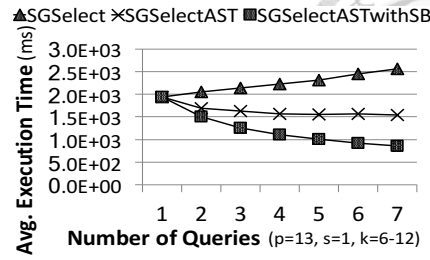
#### 4.6.1 Performance Analysis of CSGQ

Based on feedbacks from the user study, initiators are inclined to fine tune CSGQs for different recommendations. To address such an important need, we propose AST and SB to efficiently cache and index intermediate results in support of CSGQs. In the following, we compare the average query time for processing CSGQs with and without the caching mechanisms: SGSelectAST uses only AST, SGSelectASTwithSB uses both AST and SB, and SGSelect uses none of them. Figure 4.3(a) shows the results with decreasing  $k$  in consecutive queries, under varied number of consecutive queries (in  $x$ -axis), while Figure 4.3(b) shows the results with increasing  $k$ . The first query starts with  $k = 6$  and  $k = 12$  for experiments with increasing  $k$  and decreasing  $k$ , respectively. For example,  $x = 3$  with decreasing  $k$  represents the average query processing time of the queries for  $k = 12$ ,  $k = 11$  and  $k = 10$ . The results manifest that the average query processing time of SGSelectAST outperforms SGSelect because the proposed AST caches intermediate solutions for reuse and speed up the processing of succeeding queries. Recall that SBs are built to index the nodes in AST and reduce the number of nodes processed in the succeeding queries. As shown in the results, SGSelectASTwithSB leads to the greatest improvement, indicating that SB effectively avoids exploring unnecessary search space and further reduces the processing time.

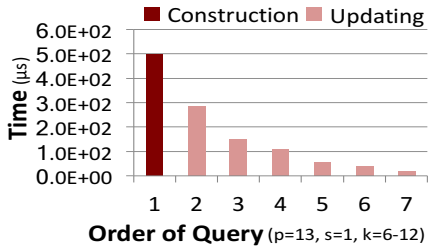
Note that, in both Figure 4.3(a) and Figure 4.3(b), the average running time of SGSelectASTwithSB consistently decreases as the number of consecutive queries increases. In contrast, the average running time of SGSelect decreases in Figure 4.3(a) but increases in Figure 4.3(b). The reason is that, for 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 later in both SGSelect and SGSelectASTwithSB. For an increasing  $k$ , the average



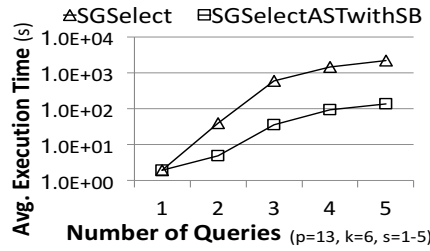
(a) Comparison of running time with decreasing  $k$ .



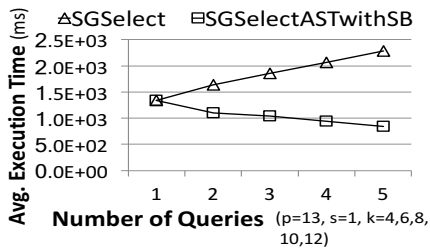
(b) Comparison of running time with increasing  $k$ .



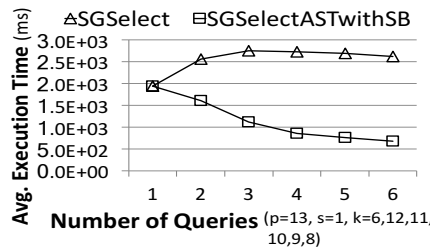
(c) Processing time of social boundary construction and updating.



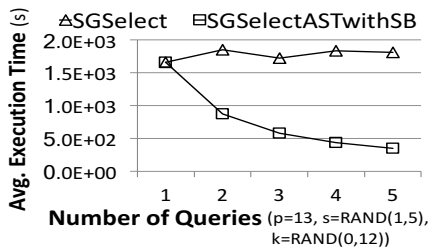
(d) Comparison of running time with increasing  $s$ .



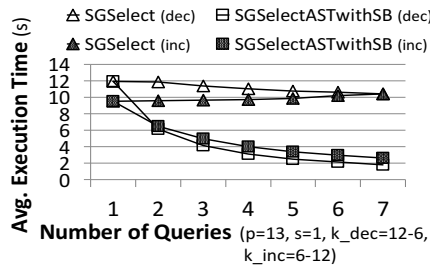
(e) Comparison of running time with faster increasing  $k$ .



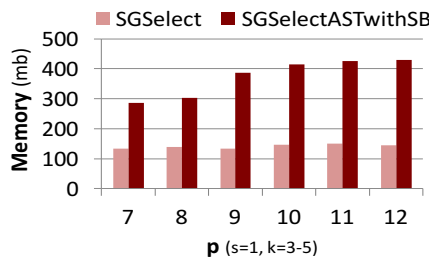
(f) Comparison of running time with dynamic  $k$ .



(g) Comparison of running time with dynamic  $s$  and  $k$ .



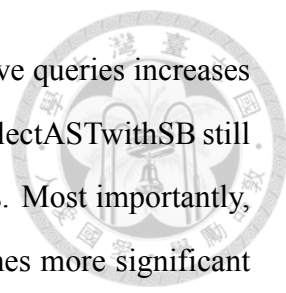
(h) Comparison of running time with various  $k$  on the YouTube dataset.



(i) Comparison of memory consumption with various  $p$ .

Figure 4.3: Experimental results of CSGQ.





running time of SGSelect becomes larger as the number of consecutive queries increases due to the relaxed constraints, while the average running time of SGSelectASTwithSB still decreases steadily with the help of the proposed caching mechanisms. Most importantly, the improvement of the running time in SGSelectASTwithSB becomes more significant as the number of queries grows. Compared with SGSelect, the query processing time of SGSelectASTwithSB is only 50% for two consecutive queries but 25% for five consecutive queries. As expected, AST and SB cache more information to effectively trim the solution space required to be explored in the coming new query.

Recall that the SBs are constructed after the first query. Moreover, after each succeeding query, it is necessary to update SBs to facilitate efficient lookup of the latest intermediate solutions in AST. Figure 4.3(c) evaluates the overhead of constructing and updating SBs. The result shows 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 construction and update of SBs may be performed offline before a new query is issued.

Figure 4.3(d) compares the running time under different settings of  $s$ . When  $s$  grows from 1 to 5, the average processing time for SGSelect is boosted by 1162 times. In contrast, SGSelectASTwithSB 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 SGSelectASTwithSB by increasing  $k$  in a faster rate in Figure 4.3(e). We further consider a more dynamic case in Figure 4.3(f), where the initiator first overly increases  $k$  from 6 to 12, and then slowly decreases it to 8. In Figure 4.3(g), the  $s$  and  $k$  of each query are randomized to simulate a case of random parameter tuning. In all the above cases, SGSelectASTwithSB always outperforms SGSelect. We also compare the algorithms in the large YouTube dataset with an increasing and decreasing  $k$  in Figure 4.3(h). 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 SGSelect-

tASTwithSB steadily becomes smaller in both cases. Moreover, the improvement over SGSelect becomes larger as the number of consecutive queries increases.

In Figure 4.3(i), we analyze the memory consumption of SGSelect and SGSelectASTwithSB under different  $p$ . For each  $p$ , there are three queries with  $k = 3, 4$  and  $5$ . SGSelect processes the three queries separately as three SGQs, while SGSelectASTwithSB processes them together as related CSGQs. Accordingly, we record the peak memory consumption of SGSelect and SGSelectASTwithSB for each  $p$ . While the memory consumption of SGSelect remains relatively steady, that of SGSelectASTwithSB becomes larger when  $p$  increases, in order to maintain AST and SBs. Although SGSelectASTwithSB requires more memory, the cached intermediate solutions effectively reduce the search space as shown in Figures 4.3(a)-4.3(h). Most importantly, these intermediate solutions only need to be cached for a short period of time during the period of activity planning.

## 4.7 Summary

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 Chapter 3, two useful queries (i.e., SGQ and STGQ) are defined to obtain the optimal set of attendees and suitable activity time. In this chapter, to support and accelerate consecutive social group queries (CSGQs), we further propose AST and SB. By effectively caching and indexing the intermediate solutions of previous queries, data lookup needed for CSGQs is facilitated. Experimental results show that, with AST and SB, the running time of CSGQs can be further reduced considerably.



## Chapter 5

# Conclusion and Future Work

In this dissertation, we study three important recommendation problems in social networks and aim to improve their efficiency. In Chapter 2, we study the link prediction problem in large-scale networks and propose a framework called Diverse Ensemble of Drastic Sparsification (DEDS), which includes various sparsification methods that are designed to preserve different measures of a network. Thus, DEDS can generate sparsified networks with significant structural differences and increase the diversity of the ensemble classifier. According to the experimental results, when a network is drastically sparsified, DEDS effectively relieves the drop in prediction accuracy and raises the AUC value. With a larger sparsification ratio, DEDS can even outperform the classifier trained from the original network. In terms of efficiency, the prediction cost is substantially reduced after the network is sparsified. If the original network is disk-resident but can fit into main memory after being sparsified, the improvement is even more significant.

Note that, in link prediction, the network is assumed to be dynamic. After a certain amount of new edges are added, the previous sparsified networks may need to be updated accordingly to preserve good prediction accuracy. A straightforward method is to sparsify the new network from the very beginning. However, since networks are often quite large, sparsifying the network all over again is not efficient. To address this issue, an incremental update algorithm for dynamic networks, which modifies the previously sparsified network instead of re-sparsifying the original network, is conceivable. With the incremental up-

date algorithm, the proposed DEDS framework will be able to generate newly sparsified networks based on old ones and save a great deal of sparsification cost.

In Chapter 3, we then study the social-temporal group query problem and its applications in activity planning. We define Social Group Query (SGQ) and Social-Temporal Group Query (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 further devise two algorithms, SGSelect and STGSelect, with various effective query processing strategies, including access ordering, distance pruning, acquaintance pruning, pivot time slots, and availability pruning. Experimental results indicate that SGSelect and STGSelect are significantly more scalable and efficient than the baseline approaches. The feedbacks from the user study further show that our approach obtains higher quality solutions with less coordination effort, thereby increasing users' willingness to organize activities.

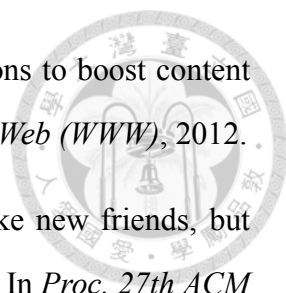
Finally, in Chapter 4, we study the consecutive group query problem. We propose Consecutive Social Group Query (CSGQ) to support a sequence of group queries. Envisaging that exploiting the intermediate solutions of previous queries may improve processing of succeeding queries, we design Accumulative Search Tree (AST) to cache the intermediate solutions of historical queries in a compact form for reuse. To facilitate efficient lookup, we further propose Social Boundary (SB) to effectively index the intermediate solutions required for processing each CSGQ with specified parameters. According to the experimental results, with the caching mechanisms, processing time of consecutive queries can be reduced considerably.

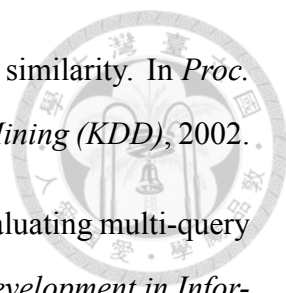
In summary, these recommendation problems we studied are important and useful, but they are also complex and computationally expensive. Nevertheless, with the diverse sparsification technique and pruning strategies devised in this dissertation, good recommendation quality can be achieved with much smaller computational cost. Therefore, the proposed framework and algorithms can be adopted in social networking websites and web collaboration tools as a value-added service, in order to efficiently provide advanced recommendation results for target users in social networks.

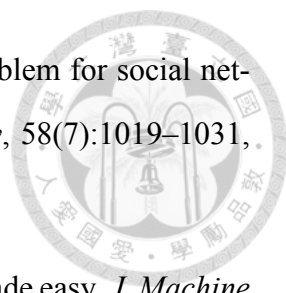


## Bibliography

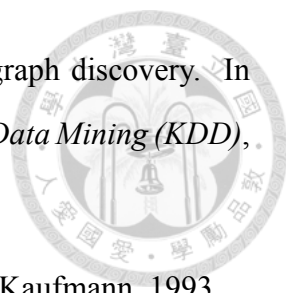
- [1] L. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230, 2001.
- [2] A. An, M. Kargar, and M. ZiHayat. Finding affordable and collaborative teams from a network of experts. In *SIAM Int'l Conf. Data Mining (SDM)*, 2013.
- [3] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2006.
- [4] B. Balasundaram, S. Butenko, and I. V. Hicks. Clique relaxations in social network analysis: The maximum  $k$ -plex problem. *Operations Research*, 59(1):133–142, 2011.
- [5] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
- [6] D. Berlowitz, S. Cohen, and B. Kimelfeld. Efficient enumeration of maximal  $k$ -plexes. In *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2015.
- [7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. 7th Int'l Conf. World Wide Web (WWW)*, 1998.
- [8] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SIAM Int'l Conf. Data Mining (SDM)*, 2004.

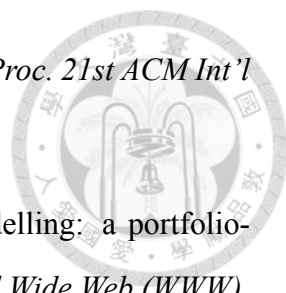
- 
- [9] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt. Recommendations to boost content spread in social networks. In *Proc. 21st Int'l Conf. World Wide Web (WWW)*, 2012.
- [10] J. Chen, W. Geyer, C. Dugan, M. J. Muller, and I. Guy. Make new friends, but keep the old: recommending people on social networking sites. In *Proc. 27th ACM SIGCHI Int'l Conf. Human Factors in Computing Systems (CHI)*, 2009.
- [11] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *Proc. 24th Int'l Conf. World Wide Web (WWW)*, 2015.
- [12] U. Feige, G. Kortsarz, and D. Peleg. The dense  $k$ -subgraph problem. *Algorithmica*, 29(3):410–421, 2001.
- [13] A. Gubichev, S. J. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proc. 19th ACM Int'l Conf. Information and Knowledge Management (CIKM)*, 2010.
- [14] B. He, Y. Ding, J. Tang, V. Reguramalingam, and J. Bollen. Mining diversity subgraph in multidisciplinary scientific collaboration networks: A meso perspective. *J. Informetrics*, 7(1):117–128, 2013.
- [15] B. Hendrickson. Chaco. In *Encyclopedia of Parallel Computing*, pages 248–249. Springer, 2011.
- [16] D. Hennessey, D. Brooks, A. Fridman, and D. E. Breen. A simplification algorithm for visualizing the structure of complex graphs. In *Proc. 12th Int'l Conf. Information Visualization (ICIV)*, 2008.
- [17] X. Hu. Using rough sets theory and database operations to construct a good ensemble of classifiers for data mining applications. In *Proc. 1st IEEE Int'l Conf. Data Mining (ICDM)*, 2001.
- [18] 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'l Conf. Management of Data (SIGMOD)*, 2014.

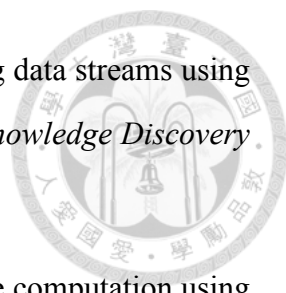
- 
- [19] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proc. 8th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2002.
- [20] E. Kanoulas, B. Carterette, P. D. Clough, and M. Sanderson. Evaluating multi-query sessions. In *Proc. 34th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, 2011.
- [21] G. Karypis. Metis and parmetis. In *Encyclopedia of Parallel Computing*, pages 1117–1124. Springer, 2011.
- [22] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.
- [23] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.
- [24] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2009.
- [25] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2006.
- [26] C. Li, P. Resnick, and Q. Mei. Multiple queries as bandit arms. In *Proc. 25th ACM Int'l Conf. Information and Knowledge Management (CIKM)*, 2016.
- [27] K. Li, W. Lu, S. Bhagat, L. V. S. Lakshmanan, and C. Yu. On social event organization. In *Proc. 20th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2014.
- [28] Y. Li, D. Wu, J. Xu, B. Choi, and W. Su. Spatial-aware interest group queries in location-based social networks. In *Proc. 21st ACM Int'l Conf. Information and Knowledge Management (CIKM)*, 2012.

- 
- [29] D. Liben-Nowell and J. M. Kleinberg. The link-prediction problem for social networks. *J. Association for Information Science and Technology*, 58(7):1019–1031, 2007.
- [30] R. Lichtenwalter and N. V. Chawla. Lpmade: Link prediction made easy. *J. Machine Learning Research*, 12:2489–2492, 2011.
- [31] R. Lichtenwalter, J. T. Lussier, and N. V. Chawla. New perspectives and methods in link prediction. In *Proc. 16th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (KDD)*, 2010.
- [32] K. Madduri and D. A. Bader. Compact graph representations and parallel connectivity algorithms for massive dynamic network analysis. In *Proc. 23rd IEEE Int’l Parallel and Distributed Processing Symposium (IPDPS)*, 2009.
- [33] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *Proc. 17th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (KDD)*, 2011.
- [34] A. Maunz, C. Helma, and S. Kramer. Efficient mining for structurally diverse subgraph patterns in large molecular databases. *Machine Learning*, 83(2):193–218, 2011.
- [35] J. J. McAuley and J. Leskovec. Discovering social circles in ego networks. *ACM Trans. on Knowledge Discovery from Data*, 8(1):4:1–4:28, 2014.
- [36] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
- [37] K. Mouratidis, J. Li, Y. Tang, and N. Mamoulis. Joint search by social and spatial proximity. *IEEE Trans. Knowledge Data Eng.*, 27(3):781–793, 2015.
- [38] M. E. J. Newman. The structure of scientific collaboration networks. *Proc. of the Nat’l Academy of Sciences USA*, 98(2):404–409, 2001.



- 
- [39] L. Qin, R. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In *Proc. 21st ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2015.
- [40] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [41] S. S. Rangapuram, T. Bühler, and M. Hein. Towards realistic team formation in social networks based on densest subgraphs. In *Proc. 22nd Int'l Conf. World Wide Web (WWW)*, 2013.
- [42] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Rev.*, 33(1-2):1–39, 2010.
- [43] 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'l Conf. World Wide Web (WWW)*, 2014.
- [44] S. B. Roy, L. V. S. Lakshmanan, and R. Liu. From group recommendations to group formation. In *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2015.
- [45] N. Ruan, R. Jin, and Y. Huang. Distance preserving graph simplification. In *Proc. 11th IEEE Int'l Conf. Data Mining (ICDM)*, 2011.
- [46] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.
- [47] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2011.
- [48] R. E. Schapire, Y. Freund, P. Barlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proc. 14th Int'l Conf. Machine Learning (ICML)*, 1997.

- 
- [49] D. Shin, S. Si, and I. S. Dhillon. Multi-scale link prediction. In *Proc. 21st ACM Int'l Conf. Information and Knowledge Management (CIKM)*, 2012.
- [50] M. Sloan and J. Wang. Dynamical information retrieval modelling: a portfolio-armed bandit machine approach. In *Proc. 21st Int'l Conf. World Wide Web (WWW)*, 2012.
- [51] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proc. 9th ACM SIGCOMM Internet Measurement Conference (IMC)*, 2009.
- [52] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *Proc. 16th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2010.
- [53] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *Proc. 18th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2012.
- [54] H. Toivonen, S. Mahler, and F. Zhou. A framework for path-oriented network simplification. In *Proc. 9th Int'l Conf. Advances in Intelligent Data Analysis (IDA)*, 2010.
- [55] C. E. Tsourakakis. The k-clique densest subgraph problem. In *Proc. 24th Int'l Conf. World Wide Web (WWW)*, 2015.
- [56] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connect. Sci.*, 8(3):385–404, 1996.
- [57] T. T. Vu, D. Song, A. Willis, S. N. Tran, and J. Li. Improving search personalisation with dynamic group formation. In *Proc. 37th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, 2014.

- 
- [58] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. 9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2003.
- [59] J. Xiang, C. Guo, and A. Aboulnaga. Scalable maximum clique computation using mapreduce. In *Proc. 29th Int'l Conf. Data Eng. (ICDE)*, 2013.
- [60] 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'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2012.
- [61] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.
- [62] S. Yang, X. Yan, B. Zong, and A. Khan. Towards effective partition management for large graphs. In *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2012.
- [63] 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'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2016.
- [64] Z. Yang, A. W. Fu, and R. Liu. Diversified top-k subgraph querying in a large graph. In *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2016.
- [65] M. Yuan, L. Chen, P. S. Yu, and T. Yu. Protecting sensitive labels in social network data anonymization. *IEEE Trans. Knowledge Data Eng.*, 25(3):633–647, 2013.



# Appendices



## A Detailed Proof

### A.1 Proof of Theorem 4.5.1

*Proof.* SGSelect is used to obtain the solution of the first SGQ, and the solution has already been proved to be optimal in Theorem 3.3.2. The search tree built in the processing of the first SGQ is taken as the initial AST and used to construct the  $(s', k')$ -SBs of different  $s'$  and  $k'$  for processing the succeeding SGQs. By induction on the sequence number  $sn$ , we will show that the theorem holds for each  $sn$ -th succeeding SGQ, where  $sn$  is a positive integer.

**Step 1.** In the following, we first prove that when  $sn = 1$  (i.e., the SB is used to solve a succeeding SGQ for the first time), processing the nodes included by the  $(s', k')$ -SB obtains the optimal solution to the succeeding SGQ with specified  $s'$  and  $k'$ . Let  $SP(s)$  denote the solution space consisting of all size- $p$  feasible solutions within  $s$  hops. With all the solution nodes and unexpanded pruned nodes from historical queries kept in AST, we are able to restore all size- $p$  feasible solutions within  $s_{old\_max}$  hops (i.e.,  $SP(s_{old\_max})$ ), where  $s_{old\_max}$  is the largest  $s$  among all processed historical queries. Note that these feasible solutions are either solution nodes or descendant nodes of unexpanded pruned nodes. That is, if we expand the cached pruned nodes to generate the solutions and compare them with the existing ones, the optimal solution can be obtained. However, an  $(s', k')$ -SB only includes a small portion of nodes in the AST. Therefore, in the following, we first prove that all the pruned nodes not included in an  $(s', k')$ -SB do not generate the optimal solution to the succeeding SGQ with specified  $s'$  and  $k'$ , and hence they do not need to reside in the  $(s', k')$ -SB. Afterwards, we prove that the internal nodes used to generate the optimal solution for a larger  $s'$  are also contained in the  $(s', k')$ -SB.

(1) Any solution node or any IU-pruned node does not need to reside in an  $(s', k')$ -SB if  $k' < U(V_S)$  or  $s' < \max_{v \in V_S} h_v$ , because the node violates the acquaintance constraint or the social radius constraint, respectively. Moreover, a solution node will not stay in an  $(s', k')$ -SB if there already exists another solution node with a smaller total social distance, since the optimal solution must have the smallest total social distance.



(2) An EE-pruned node does not need to reside in an  $(s', k')$ -SB if  $k'$  is not large enough such that

$$k' < p - |V_S| - A(V_S) + k, \quad (1)$$

and  $s'$  is also not large enough such that

$$A(V_S) < p - |V_S|. \quad (2)$$

Combining Eq. (3.1) and Eq. (1), we have

$$k' < p - |V_S| - \min_{v \in V_S} \{|V_A \cap N_v| + (k - |V_S - v - N_v|)\} + k,$$

or equivalently,

$$\min_{v \in V_S} \{|V_A \cap N_v| + (k' - |V_S - v - N_v|)\} < p - |V_S|,$$

which indicates that  $k'$  is not loose enough for this node to satisfy the exterior expansibility condition, and hence this node must be pruned again. Eq. (2) indicates that  $s'$  is too small to bring a sufficient number of new vertices, which also implies that the exterior expansibility condition is not satisfied, and hence this node must be pruned again.

(3) Similarly, an acquaintance-pruned node does not need to stay in an  $(s', k')$ -SB if  $k'$  and  $s'$  are not large enough such that

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

which means this node will be pruned again by the acquaintance pruning.

(4) A distance-pruned node does not belong to an  $(s', k')$ -SB if  $k' > k$  and  $s' = s$ , because this node will be pruned again by the solution corresponding to  $D$ . For an  $(s', k')$ -SB with  $k' < k$  or  $s' \neq s$ , the distance-pruned node does not need to be included if the updated total social distance is not sufficiently small and will be pruned by the existing

solution on the  $(s',k')$ -SB. Moreover, a distance-pruned node, as well as an EE-pruned node and an acquaintance-pruned node, do not need to be added to an  $(s',k')$ -SB with  $s' < \max_{v \in V_S} h_v$ , because these nodes violate the social radius constraint.

(5) In addition to the original pruning for each pruned node, we also examine this pruned node under the other three types of pruning strategies before it is included in an  $(s',k')$ -SB. The above examination is based on the inequalities described in Section 3.3.2 (i.e., Eqs. (3.2), (3.3), (3.4) and (3.5)), which have already been proved to trim off only infeasible or non-optimal solutions.

(6) Note that under a larger  $s'$ , new feasible solutions are necessary to be examined, i.e.,  $SP(s') \supseteq SP(s_{old\_max})$  when  $s' > s_{old\_max}$ . To bridge the gap between them (i.e.,  $SP(s') - SP(s_{old\_max})$ ), the  $(s',k')$ -SBs with  $s' > s_{old\_max}$  include the internal nodes in the AST so that the new feasible solutions can be efficiently generated by expanding the  $V_S$  in these internal nodes with newly added candidates in the  $V_A^{s'}$ . Moreover, the  $V_A^{s'}$  of pruned nodes also contain the newly included candidates within  $s'$  hops. Therefore, when the pruned nodes are successfully expanded, new feasible solutions will appear in the consequently generated sub-trees.

**Step 2.** Assume that the theorem holds when  $sn = n$  (i.e., for the  $n$ -th succeeding SGQ), which means that processing the nodes included in the  $(s',k')$ -SB obtains the optimal solution to the succeeding SGQ with specified  $s'$  and  $k'$ . After an internal node or a pruned node is successfully expanded, it is replaced by the generated sub-tree. If all of the nodes in the generated sub-trees are included in the SBs, the solution space covered by each  $(s',k')$ -SB with  $sn = n + 1$  is equal to the one with  $sn = n$ . However, to reduce the number of nodes in SBs, we use the same rules as in the SB construction to identify the newly generated nodes that can be omitted without missing the optimality. According to (1)-(6) in Step 1, processing the nodes included by the  $(s',k')$ -SB only ignores the infeasible nodes and the nodes with larger social distances. Therefore, it still obtains the optimal solution to the succeeding SGQ with specified  $s'$  and  $k'$ . Finally, by the principle of mathematical induction, the theorem holds for every positive integer  $sn$ .  $\square$

## B Pseudo Codes



---

### Algorithm 1 SGSelect

---

**Input:** Graph  $G(V, E)$ , group size  $p$ , social radius constraint with size  $s$ , and acquaintance constraint with size  $k$

**Output:** Optimal group  $F$

- 1:  $d_{q,q}^0 = 0, d_{u,q}^0 = \infty$  for  $u \neq q$ ;
  - 2: **for**  $i = 1$  to  $s$  **do**
  - 3:    $d_{q,q}^i = 0$ ;
  - 4:   **for each** vertex  $u \neq q$  in  $V$  **do**
  - 5:      $d_{u,q}^i = d_{u,q}^{i-1}$ ;
  - 6:     **for each** vertex  $v$  in  $N_u$  **do**
  - 7:       **if**  $d_{v,q}^{i-1} + c_{u,v} < d_{u,q}^i$  **then**
  - 8:          $d_{u,q}^i = d_{v,q}^{i-1} + c_{u,v}$ ;
  - 9:   Extract all vertices  $w$  in  $V$  with  $d_{w,q}^s < \infty$  and form the set  $V_F$ ;
  - 10:  $V_S = \{q\}, V_A = V_F - \{q\}, TD = \infty, D = \infty, F = \emptyset$ ;
  - 11:  $ExpandSG(V_S, V_A, TD)$ ;
  - 12: **if**  $D \neq \infty$  **then**
  - 13:   Output  $F$ ;
  - 14: **else**
  - 15:   Output “No feasible group”;
-





---

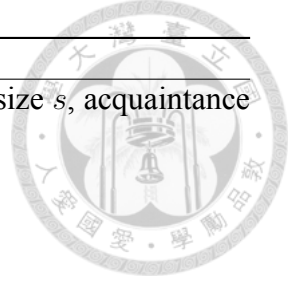
**Algorithm 2** ExpandSG

---

**Function:***ExpandSG*( $inV_S, inV_A, inTD$ )

```
1:  $V_S = inV_S, V_A = inV_A, TD = inTD$ ;  
2: while  $|V_S| + |V_A| \geq p$  do  
3:   if there is any unvisited vertex in  $V_A$  then  
4:     Select an unvisited vertex  $u$  with the minimum social distance to  $q$  and mark  $u$   
       as visited;  
5:   else if  $\theta > 0$  then  
6:     Reduce  $\theta$  and mark remaining vertices in  $V_A$  as unvisited;  
7:   else  
8:     BREAK;  
9:   if  $u$  satisfies the exterior expansibility condition then  
10:  if  $u$  satisfies the interior unfamiliarity condition then  
11:     $V_S = V_S + \{u\}, V_A = V_A - \{u\}, TD = TD + d_{u,q}$ ;  
12:    if the distance pruning condition holds for  $TD$  or the acquaintance pruning  
       condition holds for  $V_A$  then  
13:      BREAK;  
14:    else if  $|V_S| < p$  then  
15:      ExpandSG( $V_S, V_A, TD$ );  
16:    else  
17:       $D = TD, F = V_S$ ;  
18:      BREAK;  
19:    else if  $\theta = 0$  then  
20:       $V_A = V_A - \{u\}$ ;  
21:  else  
22:     $V_A = V_A - \{u\}$ ;
```

---



---

**Algorithm 3** STGSelect

---

**Input:** Graph  $G(V, E)$ , activity size  $p$ , social radius constraint with size  $s$ , acquaintance constraint with size  $k$ , and activity length  $m$

**Output:** Optimal group  $F$  and activity period  $P$

- 1:  $d_{q,q}^0 = 0, d_{u,q}^0 = \infty$  for  $u \neq q$ ;
  - 2: **for**  $i = 1$  to  $s$  **do**
  - 3:    $d_{q,q}^i = 0$ ;
  - 4:   **for each** vertex  $u \neq q$  in  $V$  **do**
  - 5:      $d_{u,q}^i = d_{u,q}^{i-1}$ ;
  - 6:     **for each** vertex  $v$  in  $N_u$  **do**
  - 7:       **if**  $d_{v,q}^{i-1} + c_{u,v} < d_{u,q}^i$  **then**
  - 8:          $d_{u,q}^i = d_{v,q}^{i-1} + c_{u,v}$ ;
  - 9:   Extract all vertices  $w$  in  $V$  with  $d_{w,q}^s < \infty$  and form the set  $V_F$ ;
  - 10:   Select a pivot time slot every  $m$  time slots, i.e., select time slots with ID as  $im$ , where  $i$  is a positive integer;
  - 11:   **while** there is any unprocessed pivot time slot  $im$  **and**  $q$  is available in  $im$  **do**
  - 12:      $V_S = \{q\}, V_A = V_F - \{q\} - \{\text{vertices not available in } im\}, TD = \infty, D = \infty,$   
       $F = \emptyset, TP = [(i-1)m + 1, (i+1)m - 1], P = \emptyset$ ;
  - 13:      $ExpandSTG(V_S, V_A, TD, TP, im)$ ;
  - 14:     Remove pivot time slot  $im$ ;
  - 15:     **if**  $D \neq \infty$  **then**
  - 16:       Output  $F$  and  $P$ ;
  - 17:     **else**
  - 18:       Output “No feasible group”;
-



---

**Algorithm 4** ExpandSTG

---

**Function:***ExpandSTG*( $inV_S, inV_A, inTD, inTP, inIM$ )

```
1:  $V_S = inV_S, V_A = inV_A, TD = inTD, TP = inTP, im = inIM$ ;  
2: while  $|V_S| + |V_A| \geq p$  do  
3:   if there is any unvisited vertex in  $V_A$  then  
4:     Select an unvisited vertex  $u$  with the minimum social distance to  $q$  and mark  $u$   
       as visited;  
5:   else  
6:     if  $\theta > 0$  then  
7:       Reduce  $\theta$  and mark remaining vertices in  $V_A$  as unvisited;  
8:     else if  $\phi < a$  predetermined threshold  $t$  then  
9:       Increase  $\phi$  and mark remaining vertices in  $V_A$  as unvisited;  
10:    if  $\phi \geq t$  then  
11:      set the RHS of the temporal extensibility condition as 0;  
12:    else  
13:      BREAK;  
14:  if  $u$  satisfies the exterior expansibility condition then  
15:    if  $u$  satisfies the interior unfamiliarity condition then  
16:      if  $u$  satisfies the temporal extensibility condition then  
17:         $V_S = V_S + \{u\}, V_A = V_A - \{u\}, TD = TD + d_{u,q}$ ;  
18:        Update  $TP$  using the available time slots of  $u$ ;  
19:        if the distance pruning condition holds for  $TD$  or the acquaintance pruning  
           condition holds for  $V_A$  or the availability pruning condition holds for  $V_A$   
           then  
20:          BREAK;  
21:        else if  $|V_S| < p$  then  
22:          ExpandSTG( $V_S, V_A, TD, TP, im$ );  
23:        else  
24:           $D = TD, F = V_S, P = TP$ ;  
25:          BREAK;  
26:        else if  $\phi \geq t$  then  
27:           $V_A = V_A - \{u\}$ ;  
28:        else if  $\theta = 0$  then  
29:           $V_A = V_A - \{u\}$ ;  
30:      else  
31:         $V_A = V_A - \{u\}$ ;
```

---