UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Problème de Sondage dans les Réseaux Sociaux Décentralisés

∴ ∴ ∴

On the Polling Problem for Decentralized Social Networks

THÈSE

présentée et soutenue publiquement le 3 Février 2015

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

HOANG Bao-Thien

Composition du jury

Président :	Nacer Boudjlida	LORIA/INRIA Nancy, Université de Lorraine
Rapporteurs :	Hamamache Kheddouci	LIRIS/CNRS, Université Claude Bernard Lyon 1
	Sébastien Tixeuil	LIP6/CNRS, UPMC Sorbonne Universités
Examinateurs :	Stéphane Frénot	CITI/INSA Lyon Telecom
	Sébastien Gambs	IRISA/INRIA Rennes, Université de Rennes 1
Directeurs de thèse :	Abdessamad Imine	LORIA/INRIA Nancy, Université de Lorraine
	Christophe Ringeissen	LORIA/INRIA Nancy

Institut National de Recherche en Informatique et en Automatique
Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Inria INVENTEURS DU MONDE NUMÉRIQUE

Loria Laboratoire lorrain de recherche en informatique et ses applications

Mis en page avec la classe thesul.

Acknowledgements

First of all, I would like to express my deepest gratefulness and sincerest thanks to my supervisors, Abdessamad Imine and Christophe Ringeissen, for their constant guidance, precious counseling, encouragement, inspiration and invaluable suggestions throughout the period of my PhD study at INRIA and Université de Lorraine. Abdessamad is a brilliant supervisor who proposed and gave me some interesting topics to work on. Since the first working days at INRIA, he has provided me everything, which I would need to succeed, with his enthusiasm tremendous patience and huge effort. My work and publications could not be achieved without his white nights, particularly the days reaching to the submission deadlines. Christophe spent his valuable time for discussing insightfully and helped me in writing clearly my problems and solutions. I also thank Michaël Rusinowitch who provided me his superior practical guidance. Michaël even is not my direct supervisor but silently encouraged me to overcome the obstacles to take the risks required for true innovation. That has been my wonderful opportunity and great honor to be their student as well as to work with them during last three years.

Moreover, it is my honor to express indispensable gratitude to all members of CASSIS team, including current and former people that I have had chance to interact with, Michaël Rusinowitch, Mathieu Turuani, Véronique Cortier, Steve Kremer, Laurent Vigneron, Walid Belkhir, Cyrille Wiedling, Huu-Hiep Nguyen, Éric Le Morvan, Mumtaz Ahmad, Houari Mahfoud, for their kind assistance, supporting and providing fun environment during my study. I also present sincere thanks to all my friends at LORIA lab, my Vietnamese friends in Nancy and many others for sharing with me the living experiences that I could not learn at school.

Additionally, and much importantly, I am greatly thankful the funding sources that helped my PhD work possible: INRIA-CORDI fellowship, ANR Streams project grant, and CNRS/Université de Lorraine grant.

Finally, but definitely not the least, I wish to contribute my thankfulness to my beloved parents and my entire family, especially thank to Ly and Bon, for their valuable loves, supports and inspirations throughout my work and life.

To my parents, Ly and Bon.

Abstract

The recent years have witnessed the explosion of online social networks (OSNs). For instance, Facebook has more than 1.32 billion active users and 829 million daily active users on average. OSN allows participants to do anything for a variety of purposes concerning business, entertainment, world's events and culture such as getting friendship, publishing and sharing information, exchanging documents, and expressing opinions in politics. Yet, in the existing OSNs like Facebook, all user data or computations on such platforms are stored and processed by the central authority that has the full knowledge and control over the network. This poses potential server failures and particularly severe privacy problems. To overcome these problems, OSN decentralization allows for users to keep control on their own data and perform computations in a distributed way without the existence of central server.

One of the current practical, useful but sensitive topic in OSNs is the polling problem. In general, polling is the way to determine the most favourite choice amongst some options from the participants. For example, one company of mobile phone has just launched a new product and may want to ask customers whether or not its features are comfortable, and user will choose one option between "Yes" or "No". Currently, there are some studies and solutions for this problem in centralized networks, such as FacebookPoll and Doodle. The challenge in studying this problem is to devise a decentralized polling protocol (without resorting for a central authority) such that it can perform a secure and accurate process to sum up the initial votes with the presence of dishonest users, who try to bias the outcome and disclose the votes of honest ones. Recently, Guerraoui *et al.* proposed polling protocols based on simple secret sharing scheme and without requiring any central authority or cryptography system. However these protocols can be deployed safely and efficiently provided that, inter alia, the social graph structure should be transformed into a ring structure-based overlay and the number of participating users is perfect square. Consequently, designing secure and efficient polling protocols regardless these constraints remains a challenging problem.

In this thesis, we are interested in polling protocols based on decentralized OSNs. More precisely, we address the problem of deploying these protocols for general social graphs and how to transform these graphs in order to increase the privacy and/or accuracy properties. Our contribution is therefore twofold.

As a first contribution, we propose three simple decentralized polling protocols that rely on the current state of social graphs. The first polling protocol uses synchronous communication model in which all connection delays are bounded, and the system is driven with the presence of global clock. Furthermore, to prevent user misbehaviours, we introduce verification procedures based on shortest path scheme and routing tables. The second polling algorithm is the enhanced version of the first one. It runs in the asynchronous network model where the arrival order of messages is unpredictable. Unlike the first protocol, we propose verification process without requiring the user's knowledge of shortest path lengths. The third polling protocol is an asynchronous one that does not require any verification procedures and contains a method for efficiently broadcasting message under a family of social graphs satisfying what we call the m-broadcasting property. We show that, despite the use of richer social graph structures, the communication and spatial complexities of this protocol are close to being linear. To evaluate the correctness of these protocols, we carry out extensive experiments under the network of 1000 nodes. The experimental result demonstrates that the dishonest coalition never affects the outcome of the poll outside the theoretical bounds.

As a second contribution, to securely perform these large-scale computations without requiring any central authority, current protocols use a simple secret sharing scheme, which enables users to obfuscate their inputs. Nevertheless, these protocols require the minimum degree of the

social graph should not be smaller than a given threshold. This condition is not often satisfied by all social graphs. In this thesis we formalize the "adding friends" problem such that we can reuse the social graphs after some minimum structural modifications consisting in adding new friendship relations. We also devise algorithms for solving this problem in centralized and decentralized networks. We validate our solution with a performance evaluation on real-world social graphs, which shows that our protocols are accurate, and inside the theoretical bounds.

Finally, the results obtained in this thesis encourage further researches of polling protocols without using some overlay structures. In future, we plan to implement our suggested polling algorithms as a plug-in over a distributed P2P social networks such as Diaspora, Friendica and Tent, and study the polling problem with the presence of Byzantine nodes. This adversary model will be stronger than the one we considered in this thesis.

Contents

Chapter 1
Introduction **1**

1.1 Context . . . 1
1.2 Motivation . . . 2
1.3 Contributions . . . 5
1.4 Roadmap . . . 7

Chapter 2
State of the Art **9**

2.1 Polling Protocols . . . 9
2.1.1 Cryptographic-based approaches . . . 9
2.1.2 Secret sharing scheme based approaches . . . 12
2.2 Graph transformation problem . . . 22
2.2.1 Matching problems . . . 23
2.2.2 Graph anonymization . . . 27
2.3 Summary and Discussion . . . 30

Chapter 3
Background **33**

3.1 Distributed systems . . . 33
3.1.1 Communication . . . 33
3.1.2 Network knowledge . . . 34
3.1.3 Timing and synchrony . . . 34
3.2 The social network model . . . 35
3.2.1 Social graph model . . . 35
3.2.2 Algorithm performance . . . 39
3.3 Preprocessing algorithms . . . 41
3.3.1 Tree building . . . 41

3.3.2 All-pairs of shortest paths and network diameter . . . 43
3.4 Summary and Discussion . . . 44

Chapter 4
Synchronous Model-based Polling Protocol 45

4.1 Polling model . . . 45
4.1.1 User behaviors . . . 46
4.1.2 Social graph model . . . 46
4.2 Polling protocol . . . 47
4.3 Correctness . . . 51
4.3.1 Properties of protocol . . . 51
4.3.2 Protocol and graph without dishonest nodes . . . 52
4.3.3 Protocol and graph with dishonest nodes . . . 54
4.3.4 Particular networks . . . 61
4.4 Experimental evaluation . . . 63
4.5 Summary and discussion . . . 65

Chapter 5
Asynchronous Model-based Polling Protocol 67

5.1 Polling model . . . 67
5.1.1 Social interactions . . . 67
5.1.2 Description of graph model . . . 68
5.2 Polling protocol . . . 69
5.3 Correctness . . . 72
5.3.1 Protocol and graph without dishonest nodes . . . 73
5.3.2 Protocol and graph with dishonest nodes . . . 76
5.4 Experimental evaluation . . . 80
5.5 Summary and discussion . . . 82

Chapter 6
Polling Protocol with Efficient Communication 83

6.1 Polling model . . . 84
6.1.1 Social graph model . . . 84
6.1.2 Secret sharing based graphs . . . 85
6.2 Polling protocol . . . 85
6.3 Correctness and Complexity Analysis . . . 88
6.3.1 Protocol and graph without dishonest nodes . . . 88

6.3.2 Protocol and graph with dishonest nodes . . . 89
6.4 Crash and message loss analysis . . . 96
6.5 Particular graphs . . . 97
6.6 Summary and discussion . . . 98

Chapter 7
On Constrained Adding Friends in Social Networks 99

7.1 Problem statement . . . 100
7.1.1 Notations . . . 100
7.1.2 Problem definition . . . 100
7.2 Centralized protocol . . . 103
7.2.1 Protocol description . . . 103
7.2.2 Correctness . . . 106
7.2.3 Greedy algorithm . . . 109
7.3 Decentralized protocol . . . 112
7.3.1 Model . . . 113
7.3.2 Protocol description . . . 114
7.3.3 Correctness . . . 116
7.3.4 Algorithms comparison . . . 118
7.4 Experimental evaluation . . . 121
7.4.1 Datasets . . . 122
7.4.2 Experimental setup . . . 122
7.4.3 Results . . . 123
7.5 Conclusion . . . 125

Chapter 8
Conclusion 127

8.1 Summary . . . 127
8.2 Discussion and future perspectives . . . 129

Appendix A
Résumé de la Thèse en Français 131

A.1 Contexte . . . 131
A.2 Motivation . . . 132
A.3 Contributions . . . 136
A.4 Contenu principal . . . 138

Bibliography **141**

List of Figures

2.1 A cluster-ring based overlay and the connection of node n. 20
2.2 SPP-Overlay sub-protocol. 21
2.3 Transform graph with only edge addition operation. 27
2.4 Examples of (a) a 4-anonymous graph and (b) a 2-anonymous graph. 29
2.5 Example of the heuristics approach of Liu and Terzi [123]. 29

3.1 Node n cannot identify whether s or v is dishonest. 38
3.2 Example of social graph. 39

4.1 Producers and consumers of node n. 47
4.2 Polling algorithm for $k = 1$. 49
4.3 A path $\omega = \langle s \equiv w_1, w_2, ..., w_{i-1}, x, w_i, ..., w_{l_0+1} \equiv n \rangle$ of length $\delta_L(s, n) + 1$. . . . 56
4.4 Misbehaviors of the dishonest nodes in the broadcasting phase. Node n receives different values emitted by the source s. (a) and (b) s and n connect together; (c) and (d) s and n are not direct friend of each other. 58
4.5 Layered networks. 62
4.6 Cluster-ring based network. 62
4.7 A circle-based network. 63
4.8 Experiment with $N = 400$, $D = 19$ to check the accuracy of protocol. 64

5.1 Polling algorithm for $k = 1$. 71
5.2 Deadlock amongst nodes $n \equiv u_1$, u_2,..., $u_{2(i-j+1)}$. 74
5.3 Experiment with $N = 1000$, $D = 50$ to check the accuracy of protocol. 81

6.1 Polling algorithm for $i = k = 1$ and $m = 3$. 87
6.2 Probability to disclose a node vote with certainty. 90
6.3 Probability to disclose a node vote without certainty. 92
6.4 A graph where protocol tolerates $\frac{m-1}{2} Diam(G)$ dishonest nodes. 95
6.5 Examples of networks satisfying the 3-broadcasting-source condition. 97

7.1 Example of value κ and λ. 102
7.2 AlgoCen algorithm example. 105
7.3 Node v links to u in A' and but not do in AlgoCen. 106
7.4 Graph where CS2 is better than CS1. 110
7.5 Graph where CS1 is better than CS2. 111
7.6 Graphs where CS2 is a $\frac{3}{2}$-approximation algorithm. 113
7.7 Example of AlgoDecen for $c = 4$ and lookahead $l = 2$. 115
7.8 Termination property in AlgoDecen. 117

7.9 Distance $\delta(u, v)$ is reduced by a factor l. 118
7.10 Two weaker nodes u and v cannot contact with each other. 119
7.11 Graph where DS2 is better than DS1. 120
7.12 Graph where DS1 is better than DS2. 121
7.13 Number of added edges. 123
7.14 Percentage of added edges in the output graph. 124

A.1 Un overlay fondé sur le graphe anneau et la connexion de nœud n. 135

Chapter 1

Introduction

Contents

1.1 Context 1
1.2 Motivation 2
1.3 Contributions 5
1.4 Roadmap 7

1.1 Context

Online social networks (OSNs) have been the most useful and typical technology of the social media in the last few years. People can discuss, exchange photos and personal news, find others of a common interest, and many more. The user number of such networks is blowing up exponentially. Just to demonstrate one typical example, as of now Facebook has more than 1.32 billion monthly active users and 829 million daily active users on average.[1] OSN allows participants to do anything for a variety of purposes related to business, entertainment, world's events and culture such as getting friendship, publishing and sharing information, exchanging documents, expressing opinions in politics.

One of the current practical, useful but sensitive topic in OSNs is *polling* process. In general, polling is the way to determine the most favorite choice amongst some options from the participants. Each participant can distribute his/her preference by voting; and after aggregating all votes, the majority option will be chosen as the final result. For instance, one company of mobile phone has just launched a new product and may want to ask customers whether or not its features are comfortable, and customers will choose one option between "Yes" or "No". Another example of polling is the scheduling that consists of a list of time options at which some tasks or events are intended to take place, and every user has to express his/her choice from those options.

Current survey showed that polling in general and scheduling in particular take an important impact in the lives of Internet users.[2] The scheduling experts frequently use electronic calendars or online scheduling services to arrange their appointments. People spend an average of 17 minutes per day on coordinating personal and business meetings. In addition, that study also revealed that online scheduling is more popular among people who mainly have business

[1] http://newsroom.fb.com/company-info/
[2] http://en.blog.doodle.com/doodle-analytics/

appointments and people who regularly schedule appointments with more than two individuals. The stats also mentioned that there is one out of five people in average use an online scheduling service, especially two-thirds of the Swiss citizens use online scheduling.

Doodle,[3] the world's largest and most successful tool for scheduling group events and voting, has over 20 million monthly users, 30K new meeting polls on daily, and 20 new Doodle polls each minute. In 2013, there are more than 17 million scheduling polls created by Doodle users. Even small groups can save about 15 minutes per poll, which means that Doodle saved more than four million hours for their users last year.

The electronic voting systems, particularly Internet voting systems, have gained popularity and have been used for government elections and referendums in the United Kingdom, Estonia and Switzerland as well as municipal elections in Canada and party primary elections in the United States and France. These systems help to facilitate decision making processes, increase participation and in some cases improve the quality of the final decision. Moreover, over the last years, a number of online trends have been well affected the outcome of the elections. Social media like OSNs can increase participation and provoke deliberation. In 2008, the presidential election in the US became as the social media election because of Barack Obama's highly effective online campaign including OSNs.

1.2 Motivation

Polling protocol. The main part we tackle in this thesis is the polling problem in OSNs. We here simply consider a binary polling with only two options "+1" or "−1" that stand for the choices "Yes" or "No" (or "Agree"/"Disagree", "Support"/"Against"). Assuming the system consists of both *honest* and *dishonest* users.[4] The goal in studying this problem is to devise a polling protocol such that it can perform a secure and accurate process to aggregate the initial votes with the presence of dishonest users, who try to bias the outcome and disclose the votes of honest ones.

The polling problem is simple but it takes an important role in incorporating user's opinion online. Thus, currently, there are some studies and solutions for this problem in two settings, centralized and distributed networks. In the centralized OSNs, like Facebook, all user data and computations on such platforms are stored and processed by the central authority which has the full knowledge and control over the network. For the particular case of polling systems, e.g., Facebook Poll[5] and Doodle, a central server is used to collect the users' votes and sum up all values to obtain output. However, this approach suffers from server failures and particularly privacy problems: each user might generally not want its vote to be known by a central entity, and it is not guaranteed the server will not bias and disclose the user votes. Furthermore, it should be noted that the centralized OSNs always consist of potential problem of uncovering user information for any commercial usage. Just to illustrate a typical example, in 2009, the Facebook's new terms of services was planed to impose in the privacy policy that the companies have an everlasting possession of personal contents even if the users already deactivate and delete their account.[6]Although the policy was finally not applied, it infers the curiousness of the companies about personal data and especially sensitive information. To overcome these shortcomings of the central authority, OSN decentralization is an alternative approach where the

[3]http://www.doodle.com/

[4]We use these terms "user", "node", "vertex", "site", and "processor" interchangeably henceforth.

[5]http://apps.facebook.com/opinionpolls/

[6]http://www.nytimes.com/2009/02/19/technology/internet/19facebook.html

users keep their own data and perform computations in a distributed way without the existence of central server.

In this thesis, we are interested in polling protocol based on decentralized OSN, where privacy of user is improved as information is not concentrated in one place. Each user site has only partial network knowledge and it is generally not possible for her/him to know the whole network and gather information or votes of other ones. In addition, we do not want to rely on cryptography for ensuring privacy or accuracy of the protocol because [82, 83]: (i) the cryptography uses complicated computation that impacts to the scalability and practicality of the protocols; (ii) all cryptography techniques rely on the assumptions which are not proved and might be broken such as the difficulty for factorizing the product of two big prime numbers or solving the discrete logarithm problem (we assume a dishonest user has bounded computation in the sense it does not have enough computing resources to break those cryptographic assumptions); and (iii) some traditional distributed computing problems can be solved without cryptography as motivated in [130, 157].

Recently, Guerraoui *et al.* [82, 83] proposed, DPol, a simple decentralized polling protocol based on secret sharing scheme (without using cryptography) where both honest and dishonest participants are considered. Honest users completely follow the assigned protocol while dishonest ones might not. All users care about their *reputation*: information related to a user is intimately considered to reflect on the associated real person. Thus, they do not want their votes to be disclosed nor their misbehaviors, if any, to be publicly exposed. In particular, dishonest users may misbehave to promote their opinion or disclose the votes of honest users. However, they never do any misbehavior which will jeopardize their reputation with probability 1. As such dishonest users are rather restricted but more reflective of the real human behavior than Byzantine users [120]. In addition, they propose some public verification procedures that are executed simultaneously with the polling algorithm to detect the misbehaviors and to expose dishonest users publicly, i.e., without disclosing the users' votes. Actually, the main goal of these procedures is to dissuade the user misbehaviors, instead of covering up the impact of dishonest users (e.g., byzantine fault tolerance BFT [36]) or to prevent their impact (e.g., cryptography based approaches [20]). The verification processes also enable honest users to tag profiles of dishonest ones based on collected testimonies. More precisely, an activity affected to the profile of the node tagged as dishonest is given. For example, when Alice is detected as a dishonest user by Bob then Alice's profile is tagged with the statement "Alice has been detected with bad behavior by Bob" and Bob's profile appears a tag "Bob accused Alice as a dishonest user". Notice that in social networks, no one would like to be tagged as dishonest by the protocol. Moreover, the authors do not consider the situation where dishonest nodes wrongfully blame honest ones (including two cases: a single user blames a group of other participants, and a group of users colluding to blame an other set of users) or attempt to spam the system with a numerous of blames, because there exist some tools or systems of social relationship between users that distinguish between legitimate and wrongful accusations. For instance, reputation systems like EigenTrust [114] and PowerTrust [193], spam mitigation systems like Ostra [138] and SocialFilter [169], and recommendation systems like SumUp [177] and Digg [2]. Most of these methods need a consensus of an unaffiliated jury to throw out the user who is considered suspicious as it tries to blame his/her friends. They also do not regard to Sybil attacks since these misbehaviors can be solved by other works such as SybilGuard [188], and SybilLimit [187].

In DPol, dishonest nodes can form a coalition so that they could get the full knowledge of the network and collaborate to achieve these goals without being detected: (i) bias the result of the poll by promoting their votes or changing the values they received from other honest nodes; (ii) infer the opinions of other nodes. In order to unify the opinions and not give compensating

effects, all dishonest nodes make the single coalition $\mathcal{D}$ of size D. However, they also want to protect their reputation from being affected. They are selfish in the sense if their accomplices have being suspected, they prefer to take care about their own reputation to covering up these accomplices.

Because of the presence of malicious users, generally, the design of typical polling system must satisfy the following main properties:

- *Privacy*: The system must protect user vote from being leaked to other ones. In other words, a dishonest node could not learn (except with negligible probability) any information from the execution of the protocol than it could do from its own input and the output of the protocol, and no way exists to deduce or verify that the vote has been cast. This property also relates to the coercion resistance of vote disclosing by examining publication of intermediate results prior to the end of a poll.

- *Accuracy*: The poll outcome must correctly reflect to the aggregation of all users' decisions. However, due to the absence of cryptography techniques that guarantee more easily the correctness of the output, we here allow some impact from dishonest nodes to the output. Thus, the protocol is said to be accurate if the difference between the output and the expected result is negligible.

DPol ensures privacy of votes and final result accuracy by limiting the impact of dishonest users. However, it has practically some disadvantages.

- Firstly, DPol relies on a structured overlay, cluster-ring-based structure inspired from [74]. Although it is efficient in term of communication cost, it is on top and really apart from the normal social graph. It does not take into account the social links among users in the sense it builds the uniform distribution of users into groups. This is not practical as we have to target a special case using notion of group instead of reserving the normal structure of the graph.

- Secondly, the number of users should be a perfect square number such that one social graph with N users is divided into $\sqrt{N}$ groups of size $\sqrt{N}$. It means the protocol could not be deployed for other social graphs which have arbitrary size.

Later, several protocols and extensions inspired from the idea of DPol have been proposed such as MPOL [65], PDP [25] and DiPA [24]. However these protocols have minor contribution compared to DPol, and they all rely on the same ring-based overlay structure.

From our literature review on the polling problem, we raise the following issues:

- **Issue 1**: *Can we devise decentralized polling protocols without cryptography and constraints such as the use of overlay structure and perfect square number of users?*

- **Issue 2**: Another issue mentioned as an enhanced version of the first one would also be interesting and brought up, that is: *Can we devise decentralized polling protocols with low communication cost?*

- **Issue 3**: We are aware that users' sites may be unreliable and crashed, and the communication channels between two sites may be lost. Therefore, the study of the effect of these factors on polling protocol is also raised: *If the network is not reliable in the sense it contains site crashes and message losses, how these factors affect the polling protocol?*

Graph transformation. Another challenge that we address concerns *graph transformation*. As introduced above, OSNs constitute live platforms exploited by huge number of users for performing large-scale computations such as conducting polls about political issues and seeking precise information on huge graph databases. To preserve data privacy during the running of these computations, recent works [65, 79, 82, 83, 181] (and our polling protocols presented in this thesis) use a secret sharing (SS) scheme. In the SS scheme, e.g., (k, n) scheme [167], a secret (e.g., cryptographic key) of a user is divided into a set of n shares before sending to n other users, such that the secret is recovered from any k or more shares in that set, but not with $k-1$ or fewer shares. Thus, this scheme allows users to obfuscate their inputs without requiring any central authority.

For instance, [24, 25, 65, 76, 82, 83] proposed distributed polling protocols in social networks without requiring cryptographic system. Instead of disclosing his/her vote, the user generates a set of shares that can regenerate the vote later, then sends only a share to each of its friends. Thus, splitting the vote into many shares enables users to protect their choices' confidentiality.

Using secret sharing based protocols imposes a threshold parameter on the number of friends of each user, e.g., the minimum degree of the social graph should be not smaller than the given threshold. Unfortunately, not all social graphs fulfill that condition. In other words, users cannot perform any common computation (e.g. polling process) if this threshold is not achieved. To satisfy the threshold condition, we try to enrich the social graph with new friendship relations. We assume these new relations will be accepted by all users as they are relevant to their common interest (i.e., performing any common computation in secure fashion). Indeed, these relationships are only for convenience. They do not enforce their partners to share any resource.

To achieve the desired graph, we use a set of graph-modification operations on the initial graph. For the sake of simplicity, we consider only edge-addition modification operation. A naive approach consists in modifying a graph in such a way that each node tries to add as many edges as possible to satisfy the threshold. Nevertheless, our main concern in this work is to answer the following question, called *adding friends* problem: *How can a graph be minimally modified to satisfy the threshold on node degree?*

The adding friends problem seems to be simple and trivial but it is really not like that. Even if we try to solve that problem in the centralized network with the presence of central server who has a global knowledge and privilege over the network, it is not intuitive to find out a suitable strategy that always gives us the optimal solution. Thus, the next issue considered here is to study the adding friends problem in the centralized networks:

- **Issue 4:** *Adding friends protocols with full knowledge (centralized approach).*

Solving the problem in the distributed system is more challenging because it is generally not possible for a node to know whole network and gather information of other nodes to request/accept friendship relations. We also tackle this problem in the decentralized setting as follows:

- **Issue 5:** *Adding friends protocols with partial knowledge (decentralized approach).*

1.3 Contributions

There are five issues to be addressed in this thesis. We respond to these challenges through presenting the following key contributions:

1. Distributed polling protocols deployed on the original social networks.

2. Centralized and decentralized adding friends protocols.

Distributed polling protocols. In designing distributed polling protocols, we consider a system model as the one in [82, 83] (presented above).

Our main objective is to keep the natural property of the graph in the sense user and social links should be preserved, and each individual can perform the voting process privately and securely without transforming the graph into any overlay structure.

First, we propose the design of simple decentralized polling protocols not requiring any central authority or cryptography system and using a secret sharing scheme. Second, we describe properties required for the social graph to ensure the correctness of each protocol. Furthermore, we cover a family of graphs and show their structures constitute necessary and sufficient conditions to ensure vote privacy and limit the impact of dishonest users on the accuracy of the polling output. It is noted that a ring-based overlay is included in our graph family.

More specifically, we describe the following three distributed polling algorithms:

- The first protocol uses *synchronous* communication model in which all connection delays are bounded, and the system is driven with the presence of global clock whose pulses must satisfy the following property: If a user n sends a message to his neighbor v at pulse p then that message must arrive at v before pulse $p+1$. Thus, the system looks like to be driven with the presence of global clock. We are aware that the data sent by one node may be corrupted by intermediate dishonest nodes. Hence, an honest node may receive distinct values of the same source. In this protocol, to prevent user misbehaviors, we introduce verification procedures based on shortest path scheme and routing tables.

- The second polling algorithm is the enhanced version of the first one. It runs in the *asynchronous* network model where no global clock exists and nodes cannot decide on their actions based on clocks. A message sent from a node to its neighbor may arrive within some finite but unpredictable time. Moreover, it is also impossible to rely on the ordering of message arrivals from different neighbors to infer the ordering of various computational events, since the order of arrivals may be arbitrary due to different message transmission speeds. Unlike the protocol above, we do not need the user's knowledge of the shortest path lengths to prevent user misbehaviors.

- Despite the use of richer social graph structures (including the ring-based structure given in [82,83]), one node can receive/send so many duplicated messages from/to other nodes. This can lead to flooding the local storage and getting high communication cost. Inspired from [166], we propose an asynchronous polling protocol which does not require any verification procedures and contains a method for efficiently broadcasting message under a family of social graphs satisfying what we call the *m-broadcasting property*. A graph satisfies the m-broadcasting property for a parameter $m \in \mathbb{N}$ such that $1 \leq m \leq d_{min}$, where d_{min} is the minimum node degree, if for each source node, there exists a topological ordering of the nodes such that every node either connects directly to the source or to some m nodes preceding it in the ordering w.r.t. the source. Accordingly, instead of accepting all messages originating from a source, a node stores only m ones passed by *ordered paths*. We show that the communication and spatial complexities of this protocol are close to be linear.

To describe carefully the distributed implementation of a polling problem, we consider the following fundamental criteria: accuracy, privacy, and the number of dishonest nodes to be

tolerated. Using the same notion of privacy parameter k in [82,83], we get the following results in a system of size N with D dishonest users for these protocols: (i) the probability that an honest node's vote is disclosed with certainty is at most $(D/N)^{k+1}$; (ii) up to $2D$ votes can be revealed with certainty by the dishonest coalition; (iii) the maximum impact from the dishonest coalition to the final result is $(6k+4)D$; (iv) the maximum number of dishonest nodes that the system can tolerate are respectively $N/10$ and $(m-1)Diam(G)/2$ (where $Diam(G)$ is the network diameter) for the first two algorithms and for the last one. We validate our solution with a performance evaluation which shows that our protocol is accurate and close to the theoretical average impact, that is $4k+2\alpha+2$, where α is the proportion of users correctly voting. Our result encourages the use of polling protocol without transforming the social graphs into other overlay structures.

Graph transformation. We first describe the adding friend problem that adds a minimum number Φ of edges to a given graph while satisfying a threshold parameter c.

For centralized social networks, we propose an algorithm computing the exact value of Φ with the time complexity in the worst case $\mathcal{O}(N^4)$ (and in the best case $\mathcal{O}(cN)$). To decrease this upper bound, we prove that there exist $\frac{3}{2}$-approximation algorithms which take time $\mathcal{O}(cN^2)$.

As for decentralized social networks, we show that no distributed algorithm is better than the centralized solution with respect to value Φ for all graph structures. In addition, we also prove that there is no best decentralized solution, i.e., any decentralized algorithm can be worse than other decentralized one for some graphs, but it can be better in some other scenarios. We validate our solution with a performance evaluation on real-world social graphs which shows that our protocols are accurate and inside the theoretical bounds. To our best knowledge, our work is the first theoretical study of the adding friend problem in centralized and decentralized models, using only a simple edge-addition operation.

Publications. Some parts of this thesis have been published and submitted to the following conferences and journals:

1. B.-T. Hoang and A. Imine. On the Polling Problem for Social Networks. In *OPODIS*, pages 46–60, 2012.

2. B.-T. Hoang and A. Imine. On constrained adding friends in social networks. In *SocInfo*, pages 467–477, 2013.

3. B.-T. Hoang and A. Imine. Flexible Polling Protocol for Decentralized Social Networks. Submitted to *ACM Transactions on Internet Technology*.

1.4 Roadmap

The rest of this thesis is organized as follows.

Chapter 2 gives a brief review of prior works related to polling problem and graph transformation. We distinguish the difference of existing works from ours as well as highlight the shortcomings and inappropriating points to apply them directly to our considering problems.

In Chapter 3, we provide some definitions for the basic model components and describe our basic concepts and notations used throughout the remaining of the thesis.

In Chapters 4 and 5, we present our polling protocols with verification procedures respectively running in synchronous and asynchronous network models. Chapter 6 represent our decentralized polling protocol that requires no verification process and can be used for a family of social graphs satisfying what we call the m-broadcasting property (where m is less than or equal to a minimum node degree). Each of these chapters introduces first the polling model, and a family of social

graphs. It then presents the polling protocol and analyzes the correctness of the protocol with and without the presence of dishonest nodes. We validate our solution with a experimental evaluation.

In addition, the polling protocols in Chapters 4 and 5 assume the existence of reliable communication among nodes. However, nodes communicate by UDP which may suffer from message loss on the communication channels or nodes' crash. In chapter 6, we analyze the effect of these factors on the protocol by considering impact on the final outcome and the probability of a node failing to decide and compute the final result.

Chapter 7 first describes our adding friends problem definition. It then presents the adding friends protocols for the centralized social networks with their correctness properties. We also describe the distributed algorithms, and show that no one amongst them is the best, as well as they cannot compete with the centralized solutions. We compare the performance of our protocols by illustrating our experimental results.

In Chapter 8, we conclude with a summary of this dissertation, discuss our achievements as well as limitations, and outline some possible future research directions.

Finally, the French version of the introduction is presented in Appendix A.

Chapter 2

State of the Art

Contents

2.1 Polling Protocols 9
2.1.1 Cryptographic-based approaches 9
2.1.2 Secret sharing scheme based approaches 12
2.2 Graph transformation problem 22
2.2.1 Matching problems 23
2.2.2 Graph anonymization 27
2.3 Summary and Discussion 30

As presented previously, we are dealing, in this thesis, with two main problems. Firstly, we tackle the problem of devising distributed polling protocols based on secret sharing and the current state of social graphs. Secondly, we address the problem of transforming optimally the social graphs in order to adapt them to the secret sharing scheme. This chapter provides a review of the evolution of some prior works related to our problems such as distributed polling protocols and graph modification. We also present their shortcomings as well as figure out several issues that those works are inadequate for our objective.

This chapter is organized as follows. Section 2.1 presents current research related to polling protocols including the approaches based on cryptography and secret sharing schemes. Section 2.2 illustrates some typical studies related to graph anonymization. Section 2.3 concludes the chapter.

2.1 Polling Protocols

Research related to distributed polling has an extremely rich state-of-the-art, aimed to ensure (i) user privacy and anonymity, i.e., users want to preserve the confidentiality of their votes, and (ii) the accuracy of the polling outcome. We divide existing research into two main categories: cryptographic based approaches and secret sharing scheme based approaches.

2.1.1 Cryptographic-based approaches

To securely perform a polling work, cryptography techniques have been used for encrypting the users' votes before collecting and decrypting the final result. More precisely, the existing protocols are based on the following methods:

MIX-net

Introduced by Chaum [37, 41], a *MIX-net* is computationally secure anonymous channel which is based on cryptography, regardless any trusted authority and allows an electronic mail system to obfuscate the content of user communication and the user identity. In this system, each site consists of a control center called *mix*. The role of a mix is to hide user in the processing, ensure no collisions amongst users, and preserve user identity. The anonymous channels are closely related to voting schemes since they all hide the correspondences between the senders and the receivers, and ballot and the voting user. Hence, Chaum [37, 41] presented an election scheme based on this technique in which each user verifies that the ballots are signed with registered users to form digital signatures. However this system has two drawbacks: (i) the length of the ciphertext delivered from the sender is very large since it is proportional to the number of mixes; (ii) the vote fairness is not guaranteed, namely, if one vote is interrupted then all other votes will be revealed. Another anonymous channel, Dining Cryptographers networks (a.k.a. *DC-nets*) [39], and an election scheme [40] based on that anonymous channel are also given by Chaum. Differing from the mix-net where user identity is preserved by the mix, each user in DC-net has its own responsibility of providing anonymization. Despite the advantage of *unconditionally secure* (which refers to the systems not relying on unproven computational hardness assumptions), DC-net still has the message collision problem and all users must share initial random numbers. To overcome these shortcomings, the authors of [147] described a secure anonymous channel which resolves the problem of the ciphertext length expansion, and an election scheme based on it which ensures the vote fairness. Other work [111] improved the robustness of MIX-net by suggesting a technique called *Randomized Partial Checking* (RPC). This technique is well suited and has already been used in binding elections in recent years. Nevertheless, Khazaei and Wikström [116] argued that RPC scheme has a serious security defect, which was consistently missed in implementations.

Robust scheme using mix-net channels was first introduced in [144] but this scheme has heavy communication cost because each node must contact regularly with others. Later some other schemes such as [7] are proposed to reduce the communication cost. In [109, 110], Jakobsson presented schemes that use mix-net channel and can efficiently do the computation for the large size of network. Nonetheless, these approaches require the existence of a verifier which has to trust at least one server to convince himself the accuracy. It means the accuracy is ensured only when the system has no conspiracy. To overcome this shortcoming, Abe [8] depicted universally verifiable schemes such that each node neither trust nor contact frequently with any server to verify the correctness, and hence the communication cost is low. Despite these enhancements, there remain limitations on the efficiency of these schemes: (i) whenever a mix is detected with misbehaviors then the mix-net must either be restarted or delayed until a replacement is found; (ii) there are not much opportunities for mixes to work in parallel.

Blind signature

Blind signature introduced by Chaum [38] is an anonymous credential, i.e., an authorization token generated by a central trusted authority and allows user to anonymously access to the system. Inspiring from this anonymous credential and RSA algorithm, the authors of [30] devised a voting protocol which enables user to anonymously vote as well as verifies the correctness and the trust level of the voting procedure. However this protocol has some shortcomings. First it is based on the anonymous credential which is known as credential sharing in the sense each user may transfer that credential to others (including dishonest users) that will use it. Moreover, dishonest users can vote more than once and affect to the final result. Thus, it assumes each user has to vote only once. Second, it relies on the presence of polling server which collects and counts the

votes of participants before informing the result. This makes this solution more closely related to centralized resolution.

Other schemes using blind signatures also take care of the problems of *fairness* and *privacy* (like the schemes based on mix-net channels). For the fairness problem, the center knows the immediate result of the computation and thus reveals the result. For the privacy problem, vote's privacy is revealed if both the administrator and the counter collude together.

For instance, the scheme in [160] solves the privacy problem, unfortunately it still consists of fairness problem. Fujioka *et al.* [71] proposed a voting scheme for large scale elections that solves both problems of fairness and privacy. Yet, voters have to send the decryption keys (with their real identifications) to the authority at the end of voting process. Hence, the security of the protocol might not be preserved if the authorities have conspired with dishonest users.

Homomorphic encryption scheme

Based on public key cryptosystems of ElGamal [75] (relying on the difficulty of computing logarithms over finite fields) and Paillier [146], Cramer *et al.* [55] presented a multi-authority secret-ballot election scheme that ensures user privacy, universal verifiability, and robustness. In that scheme, the time and communication complexities are minimal and independent of the number of authorities (comparing to other works such as [54] where these complexities are linear in the number of authorities) and each user simply gives an encrypted message containing a proof of its vote.

In addition, Benaloh and Tuinstra [22] introduced the *receipt-freeness* concept which mentions that each voting user could not show he/she has voted for a particular option and he/she could neither obtain or construct a receipt describing the content of his/her vote. Based on this concept and MIX-net channel, authors of [162] suggested a voting protocol. But the processing load is heavy due to tallying in MIX-net scheme. Other scheme using receipt-free concept and blind signatures was presented by Okamoto [145]. This scheme achieves anonymous communication but requires the activeness of user in entire polling process; this is not so practical. Hirt and Sako [94] proposed a polling protocol based on the protocols in [55, 162] and the verifier proofs in [112]. Although the vote is not disclosed by the receipt, their solution has possibility for coercion because of the so-called randomization attacks, and it also relies on unrealistic physical assumption, namely, users' misbehaviors are negligible.

Cohen and Fischer [50] showed a desirable robustness and verifiable voting scheme. Although there is high confidence in the correctness of the outcome in such a scheme, the user privacy is not ensured because of the presence of a central server which may access and compromise every user's vote. The work [23] avoided that limitation by distributing the functionality of the central server in the sense no part of server can detect the vote privacy and even if the system contains only one honest user, dishonest users could not compromise the privacy of that honest one. The accuracy of the result is obtained or verified by all users. In that scheme, however, if one of the users fails to complete the protocol properly, the entire voting process will be failed and no final result is obtained since they need the exact result.

Sako and Kilian [161] generalized the approach in [23] in such a way that it uses families of homomorphic encryptions to build some simple interactive proofs. Besides preserving privacy, and moderating communication cost, low round complexity, that approach also has other shortcoming: it violates the vote independence, i.e., users must be independent in choosing their options while voting.

Remarks. Polling protocols using cryptography help users to ensure vote privacy. However, these solutions have the following drawbacks:

- First, due to the complicated computation characteristics of cryptography, the time and message complexities of these protocols are high and impact to the scalability and practicality of the protocols. Indeed, no one could wait for hours or even days just to know the result of this simple binary polling problem.
- Second, some cryptography technique relies on the assumptions which are not proved and might be broken such as the difficulty for factorizing the product of two big prime numbers or solving the discrete logarithm problem.

In this thesis, we aim to design a polling protocol independent on cryptography. Instead of giving exactly the poll outcome, we tolerate some impact from dishonest users on accuracy but the scalability and performance are preserved.

2.1.2 Secret sharing scheme based approaches

Secret sharing schemes were introduced by both Blakley [26] and Shamir [167] independently in 1979 as a solution for safeguarding a secret (e.g., cryptography key). A (k, n) secret sharing scheme, denoted by (k, n)-*SS* and proposed by Shamir [167], is defined to be a division of a secret s into n pieces (shares) $S = \{s_1, s_2, ..., s_n\}$ $(n \geq k)$ in such a way:

(i) the secret s is recovered from any k or more pieces of S; and,

(ii) the aggregation from any $k-1$ or fewer pieces of S discloses nothing about the secret s.

This scheme is based on the polynomial interpolation, namely: given k pairs of values (x_1, y_1),..., (x_k, y_k) where all x_i are distinct, then there exists one and only one polynomial $f(x)$ of degree $k-1$ such that $f(x_i) = y_i$ for all $i \in \{1, 2, ..., k\}$. Under the assumption that s is (or could be made) a number, a $(k-1)$-degree polynomial could be chosen as $f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1}$ where $a_0 = s$. We see that by assigning $s_i = f(i)$ for $i \in \{1, 2, ..., n\}$, then the interpolation from any subset of k values of the set $\{s_1, s_2, ..., s_n\}$ can reconstruct $f(x)$ by the following formula $f(x) = \sum_{i=1}^{k} f(i) \Pi_{j=1, j \neq i}^{k} ((x - i)/(i - j))$, and the secret is obtained as $s = f(0)$. However, any subset of $k-1$ of those values does not give any information about the coefficients of $f(x)$ and, accordingly, secret s. Note that we can change value of share s_i easily without changing the original secret s by giving a new polynomial $f(x)$ with the same free term.

In the Shamir's (k, n)-SS, we call a user who divides the secret and distributes shares as *dealer*, and a user receiving the share as *shareholder*.

Later, a large number of works based on Shamir's secret sharing scheme has been published. We here introduce some typical work in this field as follows.

Secret sharing homomorphisms

In [21], Benaloh proposed secret sharing homomorphisms, a model describing a *homomorphism* property to allow multiple secrets to be combined by direct computing on shares. To understand clearly that property, let us describe the following scenario.

Assume there are two users Alice and Bob which respectively hold secrets A and B. Each one uses Shamir's (n, k)-SS [167] to distribute shares of his/her secret to n agents, such that any k agents can construct his/her secret. Then suppose that k of the agents decide that they want to determine $A + B$ while disclosing as little information about A and B as possible. How to solve this problem?

We see that each agent keeps two shares (one is of the secret A and one is of the secret B). If each of the k agents reveals the sum of the two shares it holds, each of these sums is itself

a share of the sum of the secrets $A + B$. More concisely, the sums of the shares of the secrets are shares of the sum of the secrets. It is also the case that release of these share sums gives no information about A and B other than that contained in the release of their sum $A + B$.

More general, assume each of m users keeps a "sub-secret", and the composition of these sub-secrets under some known function (e.g., sum or product) generates a "super-secret". The parties want to determine the super-secret without disclosing their sub-secrets and using cryptography. We say that a (k, n) secret sharing scheme has the homomorphism property if it satisfies the following conditions: (i) each party generates and distributes the "shares" of its sub-secret to n agents by using Shamir's secret sharing scheme (n, k) such that any k agents can construct the party's sub-secret; and, (ii) each agent can then aggregate its "shares" into a single "super-share" such that any k of the super-shares could determine the super-secret.

More formally, if we denote $\mathcal{S}$ as the domain of possible secrets, and T as the domain of legal shares. For each $I = \{i_1, i_2, ..., i_k\} \subset \{1, 2, ...n\}$ where $i_j \neq i_l$ for $j \neq l$, we define a function $F_I : T^k \to \mathcal{S}$. Then the secret s is obtained as $s = F_i(s_{i_1}, s_{i_2}, ..., s_{i_k})$. A (k, n) secret sharing scheme has the $(\oplus, \otimes)$-homomorphism property if for all I [21] we have: if $s = F_i(s_{i_1}, s_{i_2}, ..., s_{i_k})$ and $s' = F_i(s'_{i_1}, s'_{i_2}, ..., s'_{i_k})$ then $s \oplus s' = F_i(s_{i_1} \otimes s'_{i_1}, s_{i_2} \otimes s'_{i_2}, ..., s_{i_k} \otimes s'_{i_k})$.

The homomorphism property implies that the composition of the shares are shares of the composition. And it has an advantage that any k of the n agents can compute the super-secret by disclosing their super-shares, but not sharing any information related to the parties' sub-secrets. The sub-secrets can only be reconstructed if k or more agents reveal their shares.

The homomorphism property w.r.t. addition allows this scheme to be applied as a fault-tolerance method of holding verifiable secret-ballot elections. In that model, each user generates and distributes the shares of its vote to n other users in such a way that a user's vote is revealed if only if at least k shares are disclosed. Each user then aggregates the receiving shares, and sums up them to obtain the value, called *intermediate tally*. If k or more users agree to aggregate their intermediate tallies, then the final poll result is determined.

Despite the practicality of the voting application using homomorphic property, such a composition scheme of Benaloh [21] (and other schemes based on homomorphic secret sharing such as [54, 161, 165]) could lead to a significantly biased poll outcome because of the following limitations:

- First, it does not give any condition for the initial user's shares, and so, a dishonest user could generate an invalid initial set of shares (e.g., giving a set of arbitrary large values), before sending to other users.

- Second, if the network is not fully connected, the intermediate tally could be corrupted by intermediate dishonest users before aggregating at the site of honest users.

Verifiable secret sharing scheme (VSS) and Multi-party computation protocol (MPC)

In the Sharmir's (k, n) secret sharing scheme [167], any shareholder keeping the share of dealer's secret must unconditionally trust that the received share is valid. To overcome this shortcoming, Chor *et al.* [48] presented a notion of *verifiable secret sharing (VSS)*. According to this work, we say a set of n shares $\{s_1, s_2, ..., s_n\}$ is *k-consistent* if any subset of k of the n shares defines the same secret. The meaning of the "verifiability" property in this VSS is to enable shareholders to verify whether the receiving shares satisfy the k-consistent condition without disclosing the content of shares and the secret. With the presence of multiple dealers in a secret sharing scheme, the property of verifiability is more desirable since these dealers are mutually distrusted. There

are numerous applications and extensions based on VSS are proposed. In this part, we explore and discuss the relevance of some works to the design of polling protocol.

The security of VSSs can be classified into two types: computational security and *unconditionally secure*. The former term mentions the systems that assume the malicious users have limited computational resources to break a certain unproven cryptographic primitive (such as factorizing the protduct of two big prime numbers or solving the discrete logarithm problem). In contrast, the latter one refers to the systems/schemes' security not relying on unproven computational hardness assumptions. For instance, in [68], the authors introduced VSS in which they used computational security, namely, the hardness of solving discrete logarithm for the security of VSS. On the contrary, some works based on VSS addressed the unconditional security for the VSS security such as [143, 150]. We here consider only the works which use unconditional security.

Benaloh described a VSS [21] which ensures all shares are k-consistent. However, the shares may not satisfy the security requirements of a Shamir's (k, n) SS, i.e., Benaloh's VSS cannot guarantee that at least k shares are needed to reconstruct the secrets [88], and thus it could not be applied in voting.

Extending the Sharmir's SS scheme and a secret sharing scheme with mutually distrusted dealers given by Pedersen [151], Harn and Lin [88] introduced a new notion, a *strong VSS* which overcomes the limitation of Benaloh's VSS, and also depicted *a strong* (n, k, n) *VSS scheme*, where the first n is the number of dealers, k is the threshold, and the second n refers to the number of shareholders, in the SS. Liu *et al.* [124] presented *a* (n, k, n) *multi-secret sharing scheme (MSS)* to allow shareholders to share $n - k + 1$ secrets, and *a verifiable* (n, k, n) *multi-secret sharing scheme (VMSS)*, which is based on the (n, k, n) MSS. All of these schemes are unconditionally secure and are simple variation of the original Shamir's (k, n)-SS. Nonetheless, while powerful, it is difficult to apply them to polling problem, since (i) they do not give any condition for the initial users' inputs, and (ii) participants have distinct and predefined roles that may decrease the scalability and robustness of the system because the load of distributing shares belongs to the minority of users in the system (e.g., dealers) and they might be failed.

Based on VSS, other extensions and models are also proposed such as Secure Multi-Party Computation (MPC) [33, 53, 134] and Byzantine agreement [31]. The former aims to compute at each user site a function whose inputs are held by multiple distributed participants, while the latter tackles the problem of the cooperation amongst users to agree on a common data or action [120, 149]. Like VSS, some MPC and Byzantine agreement protocols make use of cryptography and address the computational security (e.g., [31, 56]), and some other MPC and Byzantine agreement protocols try to achieve unconditional security. We here focus on the latter that does not depend on any assumption about computational intractability.

Two common types of corruption from dishonest users are *passive* and *active* corruption. The passive corruption mentions that a dishonest user obtains complete information (i.e., threatens the privacy) of other honest users, but the protocol is still executed properly by any user. The latter takes place when the dishonest user has the full control of the corrupted users and users cannot follow the protocol properly.)

In [53], Cramer *et al.* showed that VSS and MPC among a set of users can efficiently be based on any linear secret sharing scheme (LSSS) for the users. LSSS is rather much weaker primitive than VSS or MPC. The authors provided an efficient construction which builds MPC and VSS protocols secure against the (passive and active) corruption from dishonest users. The approach to secure MPC in this work is general and can be applied to both the information-theoretic (i.e., secure channels) and the cryptographic setting.

Ghodosi *et al.* [78] studied unconditionally secure MPC protocols in the system consisting of passive dishonest users. They also proposed a MPC protocol in which dishonest users can corrupt up to $n-1$ users, and it makes use of a novel subprotocol for converting an additive secret sharing over a field to a multiplicative secret sharing. It is also noted that the communication cost of this MPC protocol is relatively high $\mathcal{O}(n^2)$, and this makes it inconvenient to be applied in practice.

By extending the Benaloh's SS [21], Rabin and Ben-Or [155] presented a VSS protocol by assuming there is a majority of honest nodes in the system. The secrecy achieved is unconditional and does not rely on any assumption about computational intractability. Based on VSS, they also proposed a secure MPC protocol to privately compute the user's shares and get the output with the exponentially small probability of error.

Remarks. Despite the practicality and efficiency of all applications and extensions of VSS schemes and MPC protocols presented above, they all contain some common disadvantages when applying to the polling problem:

1. All of these techniques assume the existence of fully connected network, complete synchronous communications together with a broadcast channel, secure pairwise communication channels between participants, and particularly, heavyweight computation in mathematics (that is not our objective in devising a polling protocol).

2. Some MPC protocols do not give any condition on the user's input. Thus, they allow dishonest nodes to share arbitrary data, even large values. These activities can affect the output in the potentially unbounded way.

Other later researches based on MPC have improved some aspects to limit those disadvantages. Some work [54, 55, 57, 59] focus on the scalability and usability properties in which the most relevant goal is minimizing the growth of complexity with number of nodes. Authors of [59] described an unconditionally secure protocol where the communication complexity is linear in the size n of network (and the number of dishonest users is $t < n/3$). The work in [57] presented a general MPC protocol which is simultaneously optimal, up to lower-order terms, with respect to both efficiency and fault tolerance. Ishai *et al.* [104] proposed a general MPC protocol which offers unconditional security but with minimal interaction amongst participants. Chen *et al.* [44] discussed the cost of fault tolerance in MPC complexity. They also focused on the problem of distributed computation of the function over general topologies (not only the cliques). the drawback of giving no condition of initial users' inputs still makes them difficult to be applied in the polling problem.

Anonymous Multi Party Computation (AMPC)

Malkhi and Palov introduced AMPC [131], a technique less general than Secure MPC, but sufficiently general to solve several multi-party problem models such as *oblivious transfer* (which refers to the mode of transferring information, where the sender transmits one of many pieces of information but does not know whether the receiver actually received it [29, 142, 154]), or *one-time receipt* (which refers to the protocol where user getting the receipt can verify its validity and other users can verify it at the cost of revealing the key [42, 155]). AMPC provides users with electronic anonymity without using any conventional cryptography solution, or any means of non-trivial maths. Yet this method requires the assumption about existing of secure channels between any pair of honest users and under a suitable resilience threshold of dishonest users $t < \sqrt{n}$ where n is a network size. Moreover, although this technique increases privacy, the AMPC network is built on top of normal social graph in the sense that it uses group notion to

form some levels (rows): users split their inputs into several shares and submit each share to one of the nodes in the first (top) row; the top row nodes permute the input and split into several sub-shares and send each sub-share to a node in the second level; each node in the second row combines the receiving sub-shares, permutes them and continue sending the split shares to the nodes in the third level, and so on. The process continues until all nodes in the last (bottom) level send their shares to the receiver which then aggregate these shares to get the original inputs.

Malkhi and Pavlov also presented an example application of AMPC, an *electronic voting* protocol. Despite some advantages such as the requirements of heavy computation are relaxed and users are anonymous, that protocol remains some shortcomings:

- Each voter could neither verify that the talliers have received correctly its ballots during the voting process nor confirm that it gets a valid election credential before voting. Hence, each user is not sure about the correctness of the output since it could be corrupted by other ones.
- This protocol is relatively complicated as the users have to perform the commitments on the credentials and ballots before doing computation. This leads to high communication cost for the protocol.
- This protocol is deployed on AMPC building block which is a restricted graph structure with the notion of groups. And this is not our objective as motivated in Chapter 1.

Electronic-voting protocol

Based on AMPC [131] and an extension of Rabin and Ben-Or's check vectors [155], Malkhi *et al.* [130] proposed a distributed e-voting protocol without cryptography. This protocol is based on the scheme called *enhanced check vectors*. In the enhanced check vectors protocol, the authors defined some roles for users such as *dealer* DLR, who wants to give a secret V with a meaning s, e.g., "Yes" or "No", to intermediate users (that are called *tallier* or *intermediary*) INT where s is opted from a set of possible meanings S. Then INT forwards the secret to one or more *receivers* $RCV_i, ..., RCV_m$. Each receiver should be able to independently verify whether the secret meaning is originated by DLR. However, a subset of b or fewer receivers (where b is a security parameter of the system) should not be able to reconstruct the secret V. Moreover, INT should be able to verify, with high probability, that the receivers would accept her secret.

Briefly, the e-voting protocol proceeds as follows: First, the dealer (who is also called the *voter*) generates a vote vector V that corresponds to his/her desired ballot s (from a set of possible meaning S) and is of length $b + 1$, before sending it to INT. Furthermore, it creates a set of check vectors $\{B_1, ..., B_m\}$, where m is number of potential receivers, in such a way that $VB_i = s$ for all $1 \leq i \leq m$, and one check vector for each of potential receivers. The check vectors are sent to talliers INT. INT holds a corresponding check vector B for that vote vector and verifies that indeed $VB = s$ and s is in S. The vote vectors and check vectors are also forwarded to receivers. Second, after receiving a vote vector V from INT and a check vector B for that secret, a receiver computes $s = BV$ and verifies whether s is in S. Finally, each receiver publishes all the ballots it received, and everyone can verify the election results.

Although this protocol is information-theoretically secure with strong property that the system can withstand the corruption from almost half number of nodes, it has the following important disadvantage:

- Each user needs to maintain some predefined roles such as dealer (who distributes vote vector designating her ballot), tallier (who verifies the eligibility of the vote vectors from

dealers), receiver (who receives and enables to verify the secret originated from dealer). This may decrease the scalability as a small set of participants that are not part of the system (e.g., dealers) may get the load of distributing initial shares. And the robustness of the protocol may also be decreased when that specific participants are crashed in the network.

- In addition, this protocol supports the requirement of democracy (guaranteeing that each user is able to vote), verifiability, and unconditional accuracy whereas in our objective, the polling protocol needs to be simple by relaxing such constraints.

Baudron *et al.* [20] described a distributed multi-candidate e-election system that guarantees privacy of users, provides public verification (i.e., anyone can verify and will be convinced that the voting is fair and correct), robustness (i.e., the system can tolerate with the presence of dishonest users trying to cheat during the process of voting) and receipt-freeness (to avoid coercibility and vote buying). Authors of [195] presented a light-weight scheme for electronic election. The scheme is general (that can be applied in mobile voting architecture), secure, reliable and effective. However both methods make use of cryptography. The former is based on the Paillier cryptosystem [146] and on some related zero-knowledge proof techniques [81]. The latter is inspired from the authentication protocol with revocable anonymity which combines public key cryptosystem (Merkle's puzzles [135]) and a secure secret sharing scheme [167].

Reputation (rating/ranking) schemes

There are some relations and similarities between reputation (rating) scheme and polling protocol. Generally, rating scheme relates to the situation where each user (i) evaluates the quality of other one by locally ranking that person's behavior, then (ii) aggregates the feedback (opinions) from the remaining ones of the network to compute its reputation score. A positive feedback is given if that user has good behavior and accordingly a high reputation score may be assigned. In contrast, the bad behavior results in negative opinion from other one and a low reputation score. Currently some studies and applications of rating systems in two approaches, centralized and distributed systems, are proposed. In centralized networks, a central server collects the evaluation of all users and figuring out the reputation score of each participant, e.g., e-commerce websites such as Amazon.com and Ebay.com. We here concentrate only protocols deployed on decentralized networks as follows.

Authors of [158] analyzed reputation management in the peer-to-peer (P2P) systems. They discussed the main requirements and features that a reputation system must address, and detailed some main attacks that P2P reputation system can suffer. That paper also presented some representative distributed reputation systems and showed the strong and weak points of each proposal in relation to P2P reputation system requirements. Gupta *et al.* [84] designed a mechanism for accurately rating. More specifically, they described two alternate computation schemes for the rating system such that the activity of each user is mapped to a dynamically updated reputation score: (i) *debit-credit reputation computation* (DCRC) which credits the reputation scores of partner for serving content and debits for downloading, and (ii) *credit-only reputation computation* (CORC) which credits the reputation scores of partner for serving content but does not debit for any other activities. Moreover, these schemes facilitate local storage of reputations for fast retrieval. However, it should be noted that these studies [84, 158] concentrated in the studies of ranking mechanism by either (a) analyzing specific requirements of ranking systems, or (b) presenting the technique for ranking, instead of providing efficient polling schemes.

Reputation systems are also used as mechanisms to prevent malicious behaviors [52, 60, 114] or promote collaboration [10]. Currently, content pollution is a problem in several popular P2P

file sharing networks. In [52], Costa and Almeida first presented some misbehaving scenarios for reputation systems like Scrubber and Credence [182]: (a) *collusion*: dishonest users collude to defame honest users and to increase their own reputation score, and (b) *Sybil attack*: a dishonest user acts as multiple peers to promote their corrupted objects so that the chance of that data being downloaded will be increased. And then they proposed a new hybrid peer and object reputation system to fight polluted file content.This approach, however, focuses on the collusion of users scores rather than designing an efficient voting protocol.

Trustme [168] is an approach addressing anonymous trust management through the use of mutual anonymity for both the trust host and the trust querying peer. However this approach also used public key cryptography mechanisms to provide security and prevent unwanted users. Authors of [183] presented PeerTrust which computes peer reputation scores based on three basic trust parameters (e.g., feedback a peer receives from other peers, the total number of transactions a peer performs, and the credibility of the feedback source) and two adaptive factors (e.g., transaction context factor and the community context factor). Based on the distributed hash table, EigenTrust [114] and PowerTrust [193] provided methods to compute the global reputation value of each peer by exchanging reputation through P2P network. To estimate the trustworthiness of users, TrustGuard [172] presented a mechanism which combines historical users' reputations and behavioral fluctuations. Moreover, to improve system robustness, the user reputation score is built gradually, but may be reduced quickly if a user's misbehavior is detected. In FuzzyTrust [171], it uses fuzzy logic inferences and uncertainty information to evaluate the rating of each user. Meanwhile in GossipTrust [194], a user shares its local shares by randomly distributing them to its direct neighbors until obtaining the global rationale consensus on user reputations. In [122], the authors proposed a social network based mechanism, SocialTrust, to counter collusion in the system. The rating score of each user is also considered on the impact of some social properties such as social distance, social interest, and the relationship between users. Dutta *et al.* [62] discussed a distributed rating mechanism along with rating validation schemes aimed at tackling the free-riders problem and potential collusion problem in P2P network.

Remarks. Though these works suggested methods to improve effectiveness for the reputation systems, as well as addressed the affect of user social behaviors on its rating score, it is difficult to use those solutions for solving polling problem because of the following reasons:

- All of these works have focused on the methods to count and limit the influence of collusion on user's reputation score, rather than analyzing the efficient polling protocol.
- These approaches generally do not address to user privacy as all individual ratings are public. Thus, due to the fear of retaliation from the recipient, user often avoids providing truthful opinion including negative one [156]. And thus the result of these solutions might not be correct when applying in the polling problem.
- None of the proposed schemes provide a global polling since the rating computation relies on only a subset of nodes (as a node usually contacts with only a small group of other ones in the network) [82, 83].

To overcome the lack of privacy in the previous reputation systems, inspired from [148], Hasan *et al.* proposed two decentralized privacy preserving reputation protocols, the k-shares protocol [91] and the Malicious k-shares protocol [92], which protect users privacy by hiding their initial individual opinions and disclosing only the final reputation score. However, these works still face some issues which are not our objectives when tackling the polling problem:

- Malicious users could collude and promote their opinions in such a way that each malicious one chooses other malicious neighbors for preserving its privacy. Thus they can submit many opinions which benefit for themselves without being detected.
- In [92], the protocol uses Paillier cryptosystem [146] and heavy computation for preserving privacy.

Distributed polling protocols

Guerraoui *et al.* introduced DPol [82, 83], a simple distributed polling protocol based on secret sharing scheme and without using cryptography. In the work [82], the authors considered a binary poll (i.e., a poll with two options, e.g., "Yes" and "No"), while the other work addressed to the multi-options poll (i.e., a poll with d options and each node votes for a value $v \in \{1, ..., d\}$). In DPol, each node is either honest or dishonest. Honest nodes strictly follow the protocol but dishonest ones might not. Dishonest nodes try to promote their opinion or reveal the opinion of honest nodes. However, all nodes care about their reputation in the sense that information related to a node intimately reflects on the associated real person. Dishonest nodes never misbehave if their activities might be detected with certainty. This attacker model is rather restricted and not fully Byzantine [120]. To motivate that model, they also suggested an approach that dissuades dishonest misbehaviors instead of covering up their impact (like byzantine fault tolerance BFT [36]) or to prevent their impact (like cryptography based approaches [20]). According to this approach, a verification procedure is executed with the polling algorithm, and tags the profiles of the related nodes based on collected testimonies. Moreover, DPol assumes that dishonest nodes do not wrongfully blame honest ones, and not do Sybil attacks as these kinds of attack could be solved by other existing automatic tools. In addition, dishonest nodes can form a single coalition to get the full knowledge of the network and collaborate to bias the poll output. In order to unify the opinions and not give compensating effects, all dishonest nodes make the single coalition. However, they are selfish in the sense if their accomplices have being suspected, they prefer to take care about their own reputation to covering up these accomplices.

DPol is deployed on a cluster-ring-based structure. This overlay is constructed as follows. The N nodes are clustered into $m = \sqrt{N}$ *ordered* groups, from g_0 to g_{m-1}. Each group is a clique, i.e., a node n in group g_i maintains a set $\mathcal{P}_o$ of *officemates*. Given a parameter k, a node n in group g_i also links to a fixed-size set $\mathcal{P}_p$ of nodes, called *proxies*, in the next group ($\mathcal{P}_p \subset g_{i+1 \ mod \ m}$), and a fixed-size set $\mathcal{P}_c$ of nodes, called *clients*, for which n acts as a proxy, in the previous group ($\mathcal{P}_c \subset g_{i-1 \ mod \ m}$) where $|\mathcal{P}_p| = |\mathcal{P}_c| = 2k+1$. Thus, all groups virtually form a ring with g_0 being the successor of g_{m-1} (as depicted in Fig. 2.1). Each node only receives messages that are delivered by nodes in the set $\mathcal{P}_c \cup \mathcal{P}_o$ and discards all other messages.

Briefly, DPol includes three phases: (i) voting, (ii) intermediate counting, and (iii) local tally forwarding. In the voting phase, each node generates a set of shares of its vote reflecting its votes by making use of a simple secret sharing scheme, then sends them to its clients which belong to the next group. It also receives the shares from its clients before summing them into an *individual tally*. The second phase enables a node to broadcast its individual tally all its officemates. The officemates aggregate all the individual tallies, then sum them into a *local tally*. After all the officemates have computed a local tally, each node sends the local tally of its group to its proxies. This infers each node can obtain the summing of all nodes' votes of the previous group. In the last phase, the local tallies are forwarded along the ring and all nodes finally compute the final outcome.

DPol ensures accuracy and privacy by bounding the impact on the poll outcome by dishonest nodes and balancing it with the level of privacy ensured. More precisely, in a system of N users

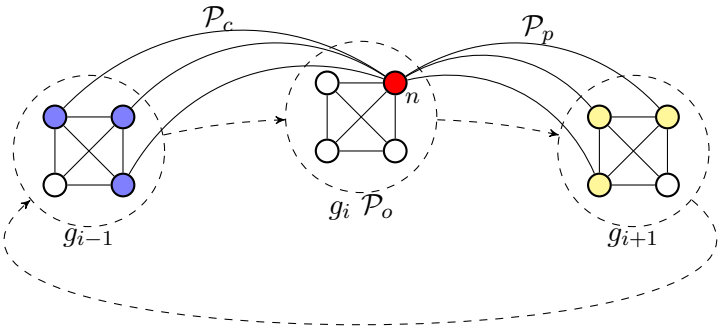


Figure 2.1: A cluster-ring based overlay and the connection of node n.

with $D < \sqrt{N}$ dishonest nodes, and a privacy parameter k, the probability for dishonest nodes to recover a node's vote is bounded by $(D/N)^{k+1}$; the impact on the polling output of binary polling is bounded by $6k+2$ (in [82]) and on the outcome of each of the d options is bounded by $3k+2$ (in [83], especially, if $d = 2$ the impact is bounded by $6k+4$). The accuracy is ensured for any assignment of nodes to groups, and the relative error (i.e., the difference between the output of the protocol and the expected poll outcome) is negligible when the network size N is large. This is due to the ability to detect with certainty the dishonest nodes who affect the result by more than $6k+2$ (in [82]) or $3k+2$ (in [83]) by using public verification procedures (i.e., detecting misbehaviors without disclosing the nodes' votes).

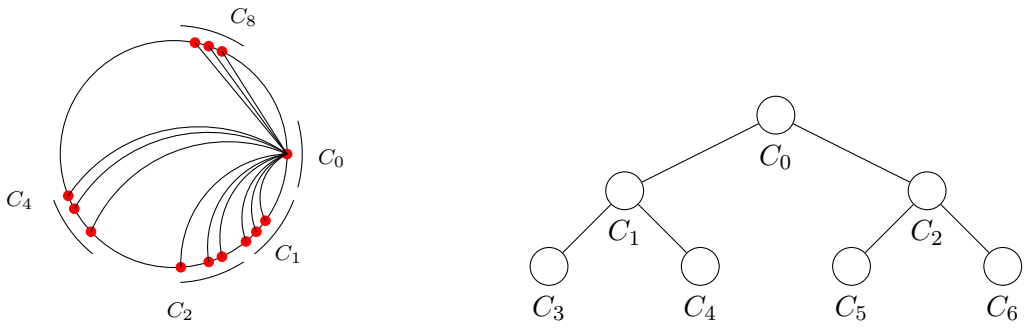
Remarks. DPol has provided a new research direction in distributed computing. However, despite strong points presented above, there remain inherent limitations of DPol when applied in practice.

- Firstly, DPol relies on a structured overlay, independent of the social graph. In spite of the efficiency in terms of communication cost, this structure is on top and really apart from the normal social graph. It does not take into account the social links among users in the sense it builds the uniform distribution of users into groups. This is not practical as we have to target a special one using notion of group instead of reserving the normal structure of the graph.

- Moreover, the number of nodes should be a perfect square such that a graph with N nodes can be divided into $\sqrt{N}$ groups of size $\sqrt{N}$. It means the protocol could not be deployed for other social graphs which have arbitrary size.

Besides, several extensions and protocols inspired from the idea of DPol have been proposed.

- PDP [25] and its extended version, DiPA [24], take into account the number of nodes splitting their inputs. More specifically, each node could cast its vote as a single share or a set of $2k+1$ shares where k is a privacy parameter (like [82, 83]). Since the votes and the shares belong to the same set (e.g. {"Yes","No"}), nodes cannot distinguish between a vote and a share. Hence, in the system of N nodes with D dishonest nodes, if not all users split their inputs, then the probability for a node to have its vote revealed with certainty by a coalition of D dishonest nodes is decrease, i.e., privacy is improved, when compared to DPol. However the impact of D dishonest participants on the final result is up to $2(k+\sqrt{N})D$. That bound is so high as we know that, theoretically, the maximum impact is $2N$, and compared to other work which impact are represented as functions of



(a) Chord overlay (b) Binary tree

Figure 2.2: SPP-Overlay sub-protocol.

value k and $k \ll N$ independent of N. Therefore, a node may obtain incorrect result even the system has a small number of dishonest nodes. DiPA, an enhanced version of PDP, presents the management of nodes failures. Nonetheless, it does not show the impact of the message loss and crash on the the accuracy as well as not compute the probability for these events occur. Additionally, these two works use the same overlay structure as DPol does.

- Authors of [65] described MPOL, an d-ary polling algorithm (i.e., poll with d options in a vote) that expands the work [83]. Comparing to DPol, this protocol has some advantages. First, it enables nodes (if all nodes are honest) to compute exact vote counts for each of the candidate instead of a winner among several candidates. Besides, MPOL allows nodes to abstain from voting, i.e., nodes are not required to vote, as long as they continue to join in other phases of the protocol. Although this mechanism could make MPOL suitable for applications (like Doodle) in which nodes may not want to express their choices but willing to support for computing the final result, it still has some drawbacks. First, to ensure accuracy of poll outcome, in MPOL, each node has to calculate the total number (called *voting counter*) of voting nodes participating in each phase of the protocol, and this value must be sent to other nodes. Thus, the final outcome is correct only if the network contains no dishonest node. Second, like DPol, this protocol relies on a ring-based structure.

Gambs *et al.* [76] introduced SPP (Secure and Private Polling), a polling protocol used in dynamic distributed networks. In an N nodes system, this protocol requires a communication complexity of $\mathcal{O}(N \log^3 N)$. The authors asserted this complexity is near-optimal by showing that (i) it is impossible to devise, with probability 1, a deterministic polling algorithm which ensures accuracy (when deployed on the system consisting of Byzantine nodes [120]) and whose communication complexity is close to be linear in N; (ii) the lower bound of the number of messages required to compute any multiparty function (i.e., global function that is a linear combination of the local inputs, and polling is a specific case) in an accurate way with high probability is at least $\Omega(N \log N)$.

Moreover, SPP guarantees the privacy of user's vote and tolerates up to $(1/2-\epsilon)N$ Byzantine nodes controlled by an adversary for any constant $0 < \epsilon < 1/2$ independent of N. Noted that Byzantine nodes can behave arbitrarily, for example by promoting their votes, revealing honest nodes' opinions, or deviating from the protocol specification. For the accuracy of the protocol, SPP assures each node outputs the exact poll outcome with high probability when the network

size N tends to infinity.

In a nutshell, SPP is composed of two sub-protocols: (i) *SPP-Overlay* provides and maintains a structured overlay (described below) when nodes leave or join the network, and (ii) *SPP-Computation* presents the polling algorithm. SPP-Overlay organizes the N nodes into g clusters, $C_0, ..., C_i, ..., C_{g-1}$, each of size $\mathcal{O}(\log N)$. The joining of nodes into the network is inspired from the protocol in [17]. These clusters are further arranged into a Chord-like overlay [186] (depicted in Fig. 2.2a) such that the nodes from cluster C_i (for $i = 0, ..., g-1$) know all the nodes from $C_{(i+2^{j-1}) \mod g}$ for all $1 \leq j \leq \log_2 g$. Furthermore, these clusters are also distributed in a binary tree of depth $\mathcal{O}(\log N)$ such that each node in cluster C_i connects to all nodes in C_{2i+1} (for $2i+1 < g-1$) and C_{2i+2} (for $2i+2 < g-1$) (see Fig. 2.2b). In the SPP-Computation sub-protocol, first each node in each cluster encrypts its input and broadcasts it to all nodes in its cluster. It then computes a *local aggregate* by adding all encrypted inputs it received from its own cluster (using the addition operation of the homomorphic cryptosystem). This local aggregate is then propagated along the binary tree as follows: starting from the clusters at the leaves of the binary tree, the nodes of these clusters send their local aggregates to all the nodes of their parent clusters. The nodes of the parent clusters computes the sum of its own local aggregate with two received ones from their children, this sum is called *partial aggregate*. The partial aggregates are also propagated towards the root. When the partial aggregates reach root cluster (e.g., C_0), they become the *global aggregate*. The nodes of the last cluster collaborate to perform the threshold decryption and to give the output.

Despite many advantages, this protocol has the following shortcomings:

- Like DPol [82, 83], SPP relies on the structured overlay of clusters. In other words, to ensure the correctness of SPP, it is necessary to construct and maintain such an overlay. The second sub-protocol, SPP-Computation, is dependent on this overlay to conduct a poll. This structure is built on top and independent of the social graph.
- SPP uses the cryptographic techniques, a threshold homomorphic cryptosytem [58].

Giurgiu *et al.* [80] defined the Scalable Secure Computing problem in a distributed social network called S^3 problem. Such computation problem is challenging as it relates to the situation that nodes need to compute a function $f : V \to U$ of their inputs in a set of constant size, in scalable, private and accurate way. Base on S^3 problem, they presented a distributed protocol, AG-S3, that computed a class of aggregation functions in a S^3 manner. On one hand side, this secret sharing scheme is suitable for polling protocol as its core has some strong characteristics such as: S^3 computation concerning a sequence of message exchanges and local computation such that honest nodes eventually get the expected final result; node is allowed to obfuscate their inputs which preserves privacy; and node can do a verification process to potentially tag the dishonest nodes' profiles. On the other hand, AG-S3 uses a special overlay graph structure like DPol where N nodes are distributed into groups of size $\sqrt{N}$. This is the same drawback as DPol we already analyzed above.

2.2 Graph transformation problem

As motivated in Chapter 1, the second challenges we address in this thesis is to transform social graphs into other ones such that users could do some secret sharing schemes on them, and thus, the user privacy and/or accuracy of protocol they are concerned are increased. There exist several different varieties of works related to this topic. We focus here on graph transformation

problem used in two hot topics: matching and graph anonymization. In this section, we examine some typical solutions for these problems. It is noted that our objective is to enrich the social graph with the minimal new friendship relations by using only edge-addition operation.

2.2.1 Matching problems

Given a graph $G = (V, E)$ where V is a set of vertex and E is a set of edges. A *matching* M in G is defined as a subset of edges in E such that no two edges have a common vertex. Based on this definition, a large amount of problems and solutions related to matching has been published.

Stable matching

In [73], Gale and Shapley introduced a *College Admissions Problem* (CAP): A set of n candidates applies for the admission of m colleges. Each college i can only accept a quota of q_i. The admission office of the college ranks the applicants in the order of his preference by evaluating their qualifications. The college can eliminate and not give proposal of admission to some applicants who do not meet certain condition even if not filling its quota. Additionally, each application also gives the ranks for the colleges in the order of his preference and excludes the ones which he does not like because of any circumstances. Under the assumption of existing no ties in the order lists, the objective of CAP is to determine an assignment of applicants to colleges, i.e., how many and which candidates will be admitted to each colleges. It is noted that each college may not know (i) whether a given applicant has also submitted the application to other colleges or he has been accepted for admission by some colleges, and (ii) the prerequisites he uses to order the ranking of colleges.

The main objective in studying this problem is to create a *stable matching* that is an assignment of applicants to colleges such that there does not exist two applicants a_1 and a_2 satisfying the following two conditions:

1. a_1 and a_2 are respectively admitted to colleges c_1 and c_2.
2. a_1 prefers c_2 to c_1, and a_2 prefers c_1 to c_2.

Gale and Shapley [73] also presented a *stable marriage problem* (SMP), a special case of college admissions problem in which the number of candidates and colleges are equal and all colleges quotas are identical: There are n men and n women in a community where each person has given ranks to all members of the opposite sex with an unique number between 1 and n in order of preference for a marriage partner. We find a reasonable solution of marrying off all members, called *stable marriages*, such that there is no pair of man and woman who would both rather be matched to each other than their current partners.

A *stable roommates problem* (SRP) [73] is similar to SMP where the users are not partitioned into two disjoint sets, and the number of users is even. More precisely, given an even number of users, and each user has a complete and strictly ordered preference list, we seek a way to divide up into pairs of roommates such that all pairs are stable, i.e., no two users who are not roommates would rather be a roommate of each other than their actual ones.

The authors of [73] presented an algorithm which solves SMP and ensures that everyone gets married, and the marriages are stable. This solution could be applied for the college admission and roommate problems. However, in the real-world setting, there might not be a stable matching for that algorithm. Irving [99] illustrated another algorithm which determines whether a stable matching exists in any instance of the problem, and if so, that matching will be found out.

The SMP and SRP are also studied in case the preference list is incomplete or contains ties (i.e., the preference list might not be strictly ordered) [12, 63, 101]. However, the problem

of finding maximum cardinality stable matching with these extensions or proving that no such matching exists is NP-hard [105, 132, 159]. Some research to give approximation algorithms of this problem have been proposed. As a stable matching is maximal matching, any two stable matchings differ in size by at most a factor of two [132]. Thus, instead of building up a polynomial-time 2-approximation algorithm, much of the sequent works addressed the improvements of the 2-approximation ratio. Iwama *et al.* [107] gave an algorithm with a performance guarantee of $(2-c/\sqrt{n})$ for a constant c. They also enhanced the approximation ratio of 1.875 in the work [106]. Later, a 5/3-approximation algorithm [118] was proposed. That is the improvement of Gale-Shapley solution where a woman may receive more than one proposal from the same man, and the roles of men and women are exchanged during the execution. Halldórsson *et al.* [86] improved the previous performance with an algorithm which approximation is expressible in terms of the number of lists with ties. These authors [87] then offered a randomized approximation algorithm of extended SMP in which the upper and lower bound of expected approximation ratios are respectively $10/7(< 1.4286)$ and $32/23$ (> 1.3913) for a restricted but still NP-hard case. Moreover, in that work [87], only preference lists of men contain ties, each man writes at most one tie, and the length of ties is two. Irving and Manlove described a polynomial-time 8/5-approximation algorithm in [102] and an extended version with the ratio of 5/3 in [103]. Iwama *et al.* [108] ameliorated the upper bound of approximation ratio to $25/17$ (< 1.4706) for the SMP with one-sided ties.

Instead of studying the ordinary stable matching problem, some works have been done in the area on restriction of it as follows. By dealing with the indifferences in the preference lists, i.e., considering the partial (rather than strictly) ordered preference lists, the notion of a stable matching could be expanded in three different ways: the *weakly*, *strongly* and *super* stable matching. A matching M is *weakly stable* if it contains no two participants $(u, v) \in E \setminus M$ such that each of them prefers being matched with the other to his partner in M. As discussed above, the problem of deciding the existence of a weakly stable matching is NP-complete for the SRP [159], and some instances of SRP admit no weakly stable matchings [85]. A matching M is *strongly stable* if there are no two participants $(u, v) \in E \setminus M$ such that u prefers v to his partner in M, and v either prefers u to his partner in M, or is indifferent between them. A matching M is *super stable* if there exist no two users u and v such that u either prefers v to his partner in M, or considers them to be tied, and v either prefers u to his partner in M, or considers them to be tied. Fleiner *et al.* [69] recently suggested an algorithm for a super-stable roommates problem.

In [9], Abraham *et al.* depicted a restriction of the SRP in which roommate pairs are ranked globally, i.e., the user's preference list may be derived from a ranking of relationship that user is involved and two roommate pairs may have the same rank. The motivation for studying this restriction is from real-world situation. For instance, it is not possible for the first-year students of the university to rank each other explicitly [16]. Alternatively, each student has to express its preference in several aspects such as the bedtime condition, the cleanliness condition, the distance between it and other ones, etc., before submitting the form describing these dimensions to get a rank. In contrast to the unrestricted SRP, weakly stable matchings in the restricted form in [9] are guaranteed to exist, and additionally, can be found in polynomial time.

Assuming the maximum roommate rank is equal to number of roommate pairs (i.e., number of edges in the graph), authors of [100, 115] defined a *rank-maximal matching* that includes the maximum possible number of rank-1 edges, and subject to this, the maximum possible number of rank-2 edges, and so on. Irving *et al.* [100] proposed an algorithm for the problem of finding a rank-maximal matching in a bipartite graph. And in [9], the authors presented a generalization of that algorithm to a non-bipartite graph.

The SMP is also approached in a distributed way where each player does not know the

complete player set and information that allow him/her to match up other ones easily. All others need to get to know each other first before they can engage together. Floreen *et al.* [70] analyzed a localized version of stable marriage where men and women are represented as vertices in a graph can only match to adjacent women or men, respectively. Each user can only exchange messages with their neighbors. The local algorithm in [70] computes an almost stable matching, namely ϵ-*stable matching*, that is a matching M in which the number of unstable edges is at most $\epsilon|M|$ where $0 \leq \epsilon < 1$. However, in this model, the set of adjacent neighbors to be matched with a given node is fixed, not dynamically changed with respect to their current network neighborhood. Other works for SMP in decentralized settings that studied similar approaches to almost stable matchings such as [66]. Moreover, an online algorithm [117], and a parallel protocol [67] are also examined. Again, they all use assumption about a fixed set of adjacent neighbors of each user.

Contrast to previous works, Arcuate and Vassilvitskii [15] considered locally stable matchings in a specialized case of stable marriage with dynamic set of matching neighbors. More precisely, they addressed a job-market game consisting of workers, and firms (who strive to hire workers). There are static *social connections*, i.e., the existing links in the original network, among workers. Each player strives to build a *matching link* to another player, i.e., a link is created and possibly deleted by the algorithm. Each firm can match to k workers, but each worker matches to only one firm. The workers are matched using a run of local variant of the Gale-Shapley algorithm. The authors also showed that best-response dynamics converge almost surely.

Hoefer [96, 97] generalized the model in [15] by extending the results on convergence of dynamics. In that model, the set of nodes is not partitioned and nodes may have more than one matching neighbor. At any point in time, each node is assumed to be matched only to other ones which are at distance not greater than some given limitation l. For instance for $l = 2$ it means that each node knows only adjacent neighbors and its neighbors of neighbors.

Remarks. Despite the advantages of the solutions of stable matching problems (including SMP, SRP, CAP) introduced above, they could not be applied in our consideration because of the following reasons:

- Most of studies give the solutions that each node has only one matching partner, for instance, a man engages with only an other woman, a student is admitted in one college. In our consideration, one node may link to more than one partner.
- Some works require the static set of matching partners in the sense a node has to match to the fixed set of friends. This differs from our concern where a set of potential neighbors to link could be changed after the creation of one connection.
- Even with the solutions with dynamic set of matching partners and it is possible for nodes to link to more than one partner, such as [15,96,97], they have the disadvantage when focusing on the maximum quality of adding edges. On the contrary, we aim to minimize the number of adding edges which quality is identical. In addition, they consider the difference between social connections and matching connections, while we drop this difference.

By defining the welfare of one node as the sum of the qualities (weights) of incident edges, Brandes and Wattenhofer [28] presented centralized and distributed algorithms in social networks which try to maximize the welfare over all nodes. However this work is different from our target. First, this work allows edge-deletion modification operation in the graph and thus, makes node much easier and flexible to choose best friends. If there exists a friend in the list which quality is worse than other one it knows (by finding, or requested by that friend), it can replace the worst

node by a better one. Second, it uses the assumption about limited number of friends of one node. For instance, in the stable state, no node can add or update more friends. There exist some nodes which degree might not satisfy our threshold constraint but they cannot request for adding new friendships since any modification could decrease their welfares. In addition, even under the assumption all edges have the same quality of 1, this method cannot solve our problem since each node will try to add as many friends as possible to maximize the number of links and this is not our objective.

Anshelevich *et al.* [13] studied the affect of social context on stability and efficiency in matching problem by incorporating social context into the decisions of users. Each user may consider the well-being of every other one to some degree.Namely, in a matching, each user obtains a reward, and also cares about the rewards of its friends. To demonstrate this care, in the matching each user holds a *perceived* or *friendship utility* which is the average rewards of the friends who are at distance between 1 and network diameter from this node. The perceived utility is the quantity users try to increase as much as possible by getting friendships (i.e., matching) with other ones. A matching is stable if it contains no pair of users which can increase the perceived utility of each one by linking to each other. Although this scheme is interesting, but again, like [28], it could not solve our problem becaues (i) it could lead to the case that one user tries to add as many friends as possible to maximize its perceived utility, and (ii) the number of friends of some user might not satisfy the threshold when it is stable and thus could not update its friend list.

b-Matching problem

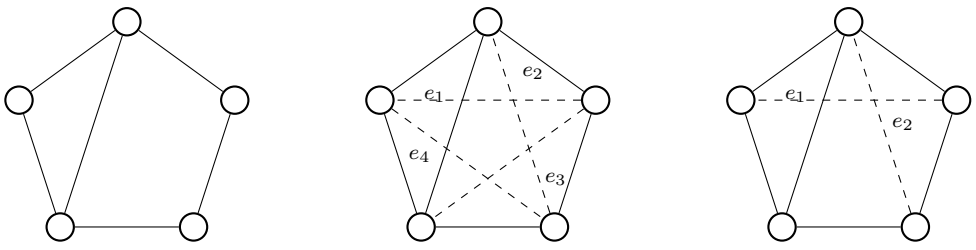
Several research related to the *b-Matching problem* have been done such as [14, 64, 119, 139, 176, 178]. The concept of b-matching defined by Edmonds [64] is as follows: For each node $n \in V$ let $\delta(n)$ denote the set of edges in E which meet n. For each vector $x = (x_e : e \in E)$ (where $x_e \in \mathbb{N}$) and a subset E' of E, let $x(E') = \sum\{x_e : e \in E'\}$. Let $b = (b_n : n \in V)$ be a positive integer vector. A *b-matching* of G is a nonnegative integer vector $x = (x_e : e \in E)$ such that $x(\delta(n)) \leq b_n \ \forall n \in V$. (A b-matching of G with $b_n = 1$ for all $n \in V$ is a matching.) From this definition, in the b-Matching problem, we have to use *edge-deletion* operation to modify G into a *subgraph* G' in such a way that each node n in the output must satisfy the condition $d_n \leq b_n$. Conversely, in our transformation problem, we aim to modify G into a *supergraph* G' by using *edge-addition* operation such that each node n has to satisfy reverse condition for degree value: $d_n \geq b_n$ (and $b_n = c \ \forall n \in V$). Moreover, the b-matching problem on the complement graph is also not our consideration as the generation of the complement graph is based on the removing and adding edges from the initial graph. These differences are also applied for the generalization of the b-Matching problem proposed by Lovász [125, 126], *General Factor problem*, where each vertex's degree of the output graph must belong to a list of possible predefined value called *degree list*. It is also known that by results in [51, 133], General Factor problem (where all degree lists may not contain gaps of length greater than one) and graph editing problem (where all degree lists are singleton) can be solved in polynomial time.

Mathieson and Szeider [133] provided a study on graph editing problems formulated as whether a given graph can be modified to satisfy certain degree constraints. More specifically, each vertex of graph has a list of numbers and the task is to edit the graph such that after editing each vertex achieves with a degree included in that list by using a limited number of editing operations such as vertex deletions, edge deletions, or edge additions. Their problems are generalization of General Factor problem [125, 126] (by allowing arbitrary lists) and Regular Subgraph Problem (when the list is singleton). Contrast to previous research on regular subgraph problems which tried to find all regular subgraphs in a given graph [27, 34, 43, 49, 127, 140, 173, 179], Math-

ieson and Szeider provided a more general setting so that in some cases the obtained graph is not a subgraph of the given one. They defined a *Weighted Degree Constraint Editing (S)* problem, or WDCE(S) for short, where S is a set of editing operations ($S \subseteq \{e, v, a\}$ where v: vertex deletion, e: edge deletion, a: edge addition), as follows: *"Given a graph $G = (V, E)$, two integers k and r, a weight function $\rho : V \cup E \to \{1, 2, ...\}$, and a degree list function $\delta : V \to 2^{\{0,...,r\}}$. Can we transform G into a graph $G' = (V', E')$ by using operations in S such that for each node $v \in V'$ we have $\sum_{uv \in E'} \rho(uv) \in \delta(v)$ with total editing cost at most k?"*.

By denoting: $\text{WDCE}_1(S)$ to indicate that the given graph is unweighted, $\text{WDCE}^*(S)$ if all degree lists are singletons, $\text{WDCE}^r(S)$ if all degree lists are $\{r\}$, and ${}_\infty\text{WDCE}(S)$ to indicate that the editing cost is not restricted, it is showed in [133] that some known problems map into WDCE(S) problem such as: *Cubic Subgraph*$={}_\infty\text{WDCE}_1^3(\{v, e\})$ [77], *r-Regular Subgraph*$={}_\infty\text{WDCE}_1^r(\{v, e\})$, and some special cases of r-Regular Subgraph problem which allow vertex deletion operation could be *Maximum Independent Set*$=\text{WDCE}_1^0(\{v\})$ [77], *Maximum Induced Matching*$=\text{WDCE}_1^1(\{v\})$ [32, 174], *Maximum Induced r-Regular Subgraph*$=\text{WDCE}_1^r(\{v\})$ [34, 141]. The General Factor problem is ${}_\infty\text{WDCE}_1(\{e\})$. It is generalization form of some simpler problems: *r-Factor problem*$=\text{WDCE}_1^r(\{e\})$ which asks whether a given graph has a spanning r-regular graph, a *Perfect Matching*$=\text{WDCE}_1^1(\{e\})$, *f-Factor problem*$=\text{WDCE}_1^*(\{e\})$, and *Perfect b-Matching*$=\text{WDCE}^*(\{e\})$. Noted that these problems relate to edge and vertex deletion operations, thus, they are different from our concerns.

Mathieson and Szeiderf [133] also showed that $\text{WDCE}^*(S)$ and $\text{WDCE}_1^*(S)$ in which S contains only edge deletion and edge addition operations, i.e., $S \subseteq \{e, a\}$, are solvable in polynomial time. With only edge addition operation (i.e., $S = \{a\}$), and a given graph G with the maximum degree Δ, they proposed a method to alter G into r-regular graph where $r = \Delta + (\Delta \mod 2)$. Although the output graph from that algorithm could be r-degree graph, and satisfied the requirements of the Issue 4 (introduced in Section 1.2 of Chapter 1), the number of adding edges is not optimal. For instance, examine a graph G with $\Delta = 3$ described in Fig. 2.3a and we desire to get an output graph where all vertices' degree are greater than or equal to 3. By applying that method, we have $r = 3 + (3 \mod 2) = 4$ and obtain a 4-regular graph which satisfies our requirements (depicted in Fig. 2.3b). However it is easy to show the existence of another method that adds only more two edges to give our desired output (see Fig. 2.3c).



(a) Initial graph G (b) 4-regular graph (c) Our desired output

Figure 2.3: Transform graph with only edge addition operation.

2.2.2 Graph anonymization

Privacy is one of the most important problems occurring in a social network. Some critical and sensitive information are disclosed due to the easy availability and due to the potential gains of statistics and analysis of that network. Generally, there are three main sorts of privacy breaches

in social networks [123]: (i) *identity disclosure*, that is user identity is revealed, (ii) *link disclosure*, that is the relationship between two users are disclosed, and (iii) *content disclosure*, the privacy extracted from data (e.g., email) is revealed. To prevent these risks of privacy, it is necessary to preprocess the user data and make it anonymously before publishing [72]. There are numerous related solutions and models have been proposed.

Firstly, it should be taken into account the privacy-preserving data mining methods. Originating the so-called k-anonymity concept for tabular data in databases [163, 164, 175], many research in the database community studied the complexity of the problem as well as tried to devise algorithms for anonymizing data records under different anonymization schemes such as [11, 113, 129, 136]. Despite the advantages of data perturbation and anonymization techniques for tabular data, these works based on k-anonymity could not solve some problems generating from graph topologies. As we know that the network structures are the sources of invaluable information for any organization wanting to recognize and learn about social groups, their dynamics and members.

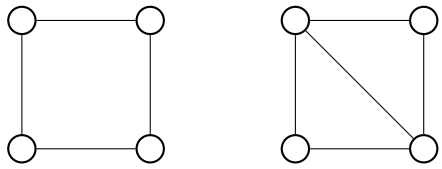
To overcome the shortcomings of privacy-preserving data mining methods, some algorithms and techniques of privacy-preserving for social networks such as graph-modification methods are presented.

The study on graph modification method to preserve identity-anonymization is originated by the result of Backstrome *et al.* [18, 19], which showed that the methods of anonymizing graph by simply removing the nodes identifiers before publishing the graph may not ensure the user privacy as dishonest nodes could infer the node identity by figuring out the restricted isomorphism problems based on uniqueness of small random subgraphs. Moreover the scheme of anonymity through structural similarity of Hay *et al.* [93], which is based on the observation that nodes having similar structures may be indistinguishable to dishonest nodes, still has problem because the data analysts can get some properties of the original graph from anonymized graph.

Inspiring the observation of structural similarity made in [93] and the concept of k-anonymity in databases [163, 164, 175], Liu and Terzi [123] presented the notation of *k-anonymous graph*, that is a graph where every node has the same degree with at least $k-1$ other ones, and the degree sequence of that graph (i.e., the multiset of positive integers corresponding to the vertex degrees in the graph) is called *k-anonymous*. Figure 2.4 depicts two examples of anonymous graphs. In Fig. 2.4a, all four nodes have the same degree, hence, the graph is 4-anonymous graph. Likewise, the graph in Fig. 2.4b is 2-anonymous one since it has two nodes of degree 2 and two other nodes of degree 3.

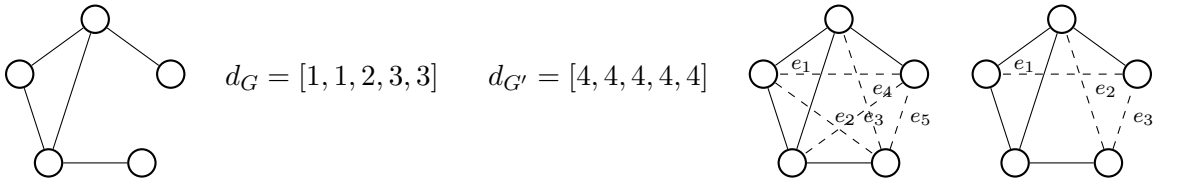
Liu and Terzi [123] also proposed a *graph anonymization* problem that, given an input graph, asks for the minimum number of edge additions and deletions to transform that graph to a k-anonymous one. In addition, a relaxed version of graph anonymization problem, called *degree anonymization* problem, with the restriction to edge additions, is also considered. That problem asks for finding a minimum-size edges set to be added into a given graph G such that the output graph is k-anonymous. This problem is known as NP-hard even for parameter $k \geq 2$ [90]. Liu and Terzi [123] suggested a heuristics approach for degree anonymization problem. Fig. 2.5 shows some steps of this approach. From the initial graph G with degree sequence $d_G = [1, 1, 2, 3, 3]$ (Figs. 2.5a and 2.5b), step 1 consists of anonymizing that degree sequence by increasing the elements in it such that each resulting element in the output degree sequence occurs at least k times (Fig.2.5c), and step 2 tries to realize the k-anonymous sequence as a supergraph of G (Fig.2.5d).

Somewhat more general models of the graph anonymization problem have been proposed later. Zhou and Pei [191, 192] studied graph anonymity related to neighborhoods of nodes: for each node there exist at least $k-1$ other nodes that share isomorphic neighborhoods. Chester



(a) (b)

Figure 2.4: Examples of (a) a 4-anonymous graph and (b) a 2-anonymous graph.



(a) Input graph G with $k = 3$ (b) initial degree sequence (c) k-anonymized degree sequence (d) k-anonymous graph G' (e) Our desired output

Figure 2.5: Example of the heuristics approach of Liu and Terzi [123].

et al. [46] investigated the variant of nodes to be added (instead of edges). Originating from the research in [189] that some users in a social network might agree to allow some information release while others want them to be anonymous, Chester *et al.* [45] introduced a problem of *subset anonymization* for an unweighted graph. An other model of subset anonymization, *k-label sequence subset anonymity problem*, for a weighted graph are also consider in [47].

Remarks. Although these graph anonymization problems and their solutions of graph transformation enable the anonymity for nodes in the network, but they all are different from our consideration because: (i) they focus on the number of appearance of a degree value in the degree sequence, and (ii) there is no threshold constraint on the value of node's degree. For instance, from the input graph in Fig. 2.5a, we can have the output graph satisfying the requirements that each node degree is at least 3 (introduced by the Issue 4 in Section 1.2 of Chapter 1) by adding more three edges (Fig. 2.5e) instead of adding five edges like Fig. 2.5d. Note that these differences also apply to some enhanced works of [123] such as [89] (which improved the upper and lower bounds heuristics), [47] (which improved complexity), [128] (which showed a more efficient greedy algorithm) and [35] (which presented an algorithm for k-degree anonymity on large networks).

Not only protection of identities for users in social networks, the protection of relationship between users is also vital. To limit the possibility of sensitive connection disclosure, Zheleva and Getoor [190] presented five strategies based on edge-deletion and node-merging operations. Ying and Wu [185] studied the affect of randomly adding and removing edges on the graph properties and graph anonymization. More specifically, they focused on the eigenvalues of a network and presented a spectrum preserving graph randomization method that preserves network properties and protecting edge anonymity. Despite the solutions could help anonymization, these two works' solutions use edge-deletion and do not allow the limitation of node degree which are our concerns.

Approaches	Create an invalid set of shares	Intermediate tally are unlimitedly corrupted	Rely on overlays or restricted graph structures	Use crypto or heavy computation	Existence of fully connected network	Existence of secure channels	Preserve user privacy	Define user's role
Secret sharing homomorphisms	Yes	Yes	No	No	No	No	Yes	No
VSS and MPC	Yes	No	No	Yes	Yes	Yes	Yes	No
AMPC	No	Yes	Yes	No	No	Yes	Yes	No
E-voting without crypto	No	No	Yes	No	No	Yes	Yes	Yes
Reputation scheme	Yes	Yes	No	No	No	No	No	No
DPol, MPOL, DiPA, PDP, AG-S3	No	No	Yes	No	No	No	Yes	No
SPP	No	No	Yes	Yes	No	No	Yes	No

Table 2.1: Comparison of approaches based on secret sharing schemes when applied for polling problem.

Approaches	Allow only static matching partners	Give threshold constraints for node degree	Use edge-deletion operation	Use only edge-addition operation	Focus on the number of appearance of a degree value	Ensure the threshold condition
CAP, SMP, SRP	Yes	No	No	Yes	No	No
Maximize welfare [28]	No	No	Yes	No	No	No
***b*-Matching**	No	Yes	Yes	No	No	Yes
$WDCE_1^*\{a\}$	No	Yes	No	Yes	No	Yes
Graph anonymization	No	No	No	No	Yes	No

Table 2.2: Comparison approaches for the graph transformation problem. Note that with $WDCE_1^*\{a\}$ we consider only the method to alter a given graph into a regular graph.

2.3 Summary and Discussion

From what we have surveyed, we can see that existing works are inadequate to our requirements for polling and graph transformation protocols.

For the polling problem, the previous approaches are either mainly based on cryptography techniques, or based on secret sharing schemes. The former is not our target since we would like to address the simple polling protocol without using complicated computation. In the latter, much research have been proposed, however, they all have some drawbacks when deployed for polling problem such as creating an invalid initial set of shares, the intermediate tally could be corrupted by dishonest users, not taking into account the user privacy, predefining some roles for users, and especially relying on some overlay structures. Table 2.1 resumes the differences amongst the approaches based on secret sharing schemes. On the contrary, our aim is to design simple distributed polling protocols that use no cryptography as well as rely on the natural property of the graph in the sense the nodes and social links of the graph should be preserved, and each individual can perform the voting process privately and securely on these graphs, not on any overlay structures.

For the graph transformation problem, most of the recent approaches related to matching or graph anonymization problems that either allow only static matching partners for one node (e.g., SMP, SRP), or modify a graph into a subgraph such that each node degree of the output must satisfy an upper bound condition, or do not give any threshold constraint for node degree such that the node can add more friends to maximize the welfare (or it cannot add more friends because it is already stable w.r.t. welfare), or focus on only the number of appearance of the degree value (in the list of all nodes' degrees). Table 2.2 summarizes the comparison of the graph transformation approaches presented in this chapter. Conversely, our objective is to transform a graph G into a supergraph G' by using only edge-addition operation such that each node degree

of the output must satisfy a lower bound condition.

In the following Chapter, we define some terms, notions, basic model components and preprocessing procedures that will be used throughout this thesis.

Chapter 3

Background

Contents

3.1 Distributed systems 33
3.1.1 Communication 33
3.1.2 Network knowledge 34
3.1.3 Timing and synchrony 34
3.2 The social network model 35
3.2.1 Social graph model 35
3.2.2 Algorithm performance 39
3.3 Preprocessing algorithms 41
3.3.1 Tree building 41
3.3.2 All-pairs of shortest paths and network diameter 43
3.4 Summary and Discussion 44

This chapter provides some definitions for the basic model components and computational notations in a distributed system that will be used throughout this thesis. We also present the preprocessing procedures needed for the remaining algorithms in this thesis. The content of this chapter is inspired from the book [152].

This chapter is organized as follows. Section 3.1 illustrates some issues in the distributed system. We give all terms and notations as well as present a social network that will be used throughout this thesis in Section 3.2.1. Then Section 3.3 describes some protocols to be used as preprocessing procedures for others algorithms later on. Finally, Section 3.4 concludes the chapter.

3.1 Distributed systems

This section presents some issues in the distributed setting including communication information, partial knowledge of node, timing and synchrony. Noted that these issues are introduced in the book [152].

3.1.1 Communication

All processors in distributed systems communicate to each other by exchanging information. This aspect is implicitly nonexistent in centralized networks. However the requirements for

characterizing, developing and analyzing methods for communication amongst participants in the distributed system are necessary since the communication has certain costs associated with it. In addition, there are some other properties related to communication such as speed limitation of the information transmission, or amounts of exchanged information. In certain situations, some costs dominate others; for instance, with the high development of processing speed of processors, local processing at the sites of the system may be negligible compared to the real communication cost associated with sites. Within the context of this thesis, we consider only the communication cost to propose efficient solutions.

3.1.2 Network knowledge

The knowledge of the network topology is one of the most different points between centralized setting and decentralized setting. In the former one, each processor has a full knowledge of network structure as well as all information needed for carrying out the computation. In particular, the input is known in its entirety to the processor at the initiation of the computation. All intermediate data or result obtained during that computation is also informed and available for all processors.

In contrast, a participant in the distributed network has only a partial information of the system and ongoing activities in the sense it may lack some critical information for doing the computation since it is hidden. For instance, each site may not know the entire input of the problem at hand, which may be physically distributed among the sites, but only some partial information stored locally. Moreover, it is not ensured or has any information about the participation or collaboration of other sites for the necessary computation as well as the origin of the information, the current stage and situation of other participants. Thus the coordinating of the joint activities amongst processors on a common task is much more difficult.

Furthermore, a processor does not necessarily know much information about the surrounding ones. This concerns to that fact that the amount of topological knowledge available at each site is limited. For instance, *anonymous network* [184] is the particular restricted model where all sites are indistinguishable, not identified by labels and know nothing on the topology of the network. The mostly powerful model assumes the complete topological knowledge of the network is maintained at every processor. In this thesis, we consider two somewhat more realistic models: (i) a model in which processors are uniquely identified and each processor knows the identities of its neighbors, and also the number of processors in the system, its diameter, or its maximum degree; (ii) a more permissive model which allows a processor to know other ones at some neighborhood.

3.1.3 Timing and synchrony

In the distributed system, the *time* truly affects to the development of the system as well as analyzing problems and characterizing solutions. Hence, the concept of time and its systematic methodologies are the most active research areas in the domain of distributed computing. *Synchrony* of the system is the typical notation in this context. Based on the level of the synchrony, two main models are classified: the (fully) synchronous model and asynchronous model.

The synchronous model. In this model, it is assumed that all connection delays are bounded. Each processor keeps a local clock whose pulses must satisfy the following property: If a processor n sends a message to its neighbor v at pulse p then it must arrive at v before pulse $p + 1$ of v. For this consideration, the system looks like to be driven with the presence of global clock. Each cycle of a processor might be composed of the following three steps:

1. Send messages to the neighbors.
2. Receive messages from the neighbors.
3. Perform some local computation.

We assume the processing and computing procedures at each local site take negligible time compared to message transmission. Therefore, each processor spends the entire cycle on waiting for receiving messages from its neighbors at the start of the cycle.

The asynchronous model. In this model, the system contains no global clock and processors cannot decide on their actions based on clocks. A message sent from a processor to its neighbor may arrive within some finite but unpredictable time. In other words, clocks are rather useless, at least as far as communication is concerned. We cannot rely on the waiting time elapsed to deduce that a neighbor did not send a message by a certain time or that the message was lost during transmission; it can always be the case that the message is still on its way. Moreover, it is also impossible to rely on the ordering of message arrivals from different neighbors to infer the ordering of various computational events, since the order of arrivals may reverse due to different message transmission speeds.

Besides these two models, some other models are proposed in which the systems are attempted in certain limited degree of synchrony. For example, each processor in the network has upper and lower bounds on message transmission times. Both synchronous and asynchronous models are not realistic than those models but they are useful for studying the behavior of a variety of problems and issues, as they help to define and limit their potential behavior in intermediate models. For instance, a lower bound or an impossibility result proven for a certain problem in the fully synchronous model will be extremely useful as it applies to every other intermediate model as well. Likewise, if an algorithm operated in the fully asynchronous model implies that an algorithm of the same complexities (or better) exists for every other model.

3.2 The social network model

This section first presents the definitions for the basic social graph model components, then introduces complexity measures. Finally we discuss the representative models one can employ in studying the behavior of the distributed systems.

3.2.1 Social graph model

Throughout this thesis, we present the social network in our problem in the form of models of social graphs. In this part, we first describe the system components, and then define the terms and notations of graph parameters which occur intensively later on.

Model

Definition 3.1 (Graph)**.** *The system model consists of a point-to-point communication network, described by a simple undirected connected social graph $G = (V, E)$ with $N = |V|$ uniquely identified nodes representing users (or network processors) and the set E of edges representing bidirectional social links.*

The unique identifiers of nodes are assumed to be taken from an ordered set of integers $P = \{p_1, p_2, ...\}$, where $p_i < p_{i+1}$ for every $i \geq 1$, and the one-to-one mapping between each node

and its identifier is $Id : V \to P$. (We sometimes ignore the distinction between the node itself and its identifier, i.e., we may refer to the node n as either n or $Id(n)$ interchangeably, where no confusion arises.) These identifiers may be fixed and assigned to the nodes on the hardware level. Additionally, we also assume that the identifier $Id(n)$ of each node n is of $\mathcal{O}(\log N)$ bits.

The node set of a graph G is denoted by $V(G)$ and the edge set by $E(G)$. For the sake of simplicity, when G is clear from the context, we sometimes omit (G) and write the notations in the forms of abbreviations without that script, e.g., $V(G)$ and $E(G)$ are written as V and E.

Definition 3.2 (Adjacency)**.** *An edge (u, v) is said to* join *the node u to the node v and is denoted by uv. We also say that u and v are* adjacent *nodes and they are* incident *with the edge uv. Two distinct edges with a common endnode are* adjacent*.*

We also use a function $e(u, v)$ to indicate the adjacency between two nodes u and v, namely, $e(u, v) = 1$ if u is adjacent to v, and $e(u, v) = 0$ otherwise.

Definition 3.3 (Supergraph and subgraph)**.** *A graph $G' = (V', E')$ is a* subgraph *of a graph $G = (V, E)$ and denoted by $G' \subset G$ if $V' \subset V$ and $E' \subset E$. We also say that G is a* supergraph *of a graph G'.*

Definition 3.4 (Neighbor of a node)**.** *The set of nodes adjacent to a node $n \in V$ is called (direct)* neighbors *(or* friends*) of n and denoted by $\Gamma(n)$, and $d_n = |\Gamma(n)|$ said to be the* degree *of n.*

The *minimum degree* and *maximum degree* of the node are respectively denoted by $d_{min}(G)$ and $d_{max}(G)$ (or d_{min} and d_{max}). If $d_{min}(G) = d_{max}(G) = r$, i.e., every node of G has degree of r, then G is said to be *r-regular*.

Attacker model in the polling protocols

We consider the same attacker model as the one given by Guerraoui *et al.* [82, 83]. The system is composed of *honest* and *dishonest* nodes. Honest nodes completely comply with the assigned protocol and take care about their privacy while dishonest ones might not. All nodes have to send/receive/forward messages if they are requested. All dishonest nodes can form a coalition to get the knowledge of the network and try to achieve these goals without being detected: (i) bias the result of the election by promoting their votes or changing the values they received from other honest nodes; (ii) infer the opinion of other nodes. In order to unify the opinion and not give compensating effects, all dishonest nodes make the single coalition $\mathcal{D}$ of size D and give the same corrupted values.

In order to avoid using cryptography, we exploit the social nature of the nodes, especially the one-to-one association between social network identities and real identities, in the attacker model. More specifically, all nodes in social networks care about their *reputation*: any information related to a node intimately reflects on the associated real person. Using the concern of reputation, we propose an approach to dissuade misbehaviors: each node can execute a verification procedure to detect dishonest behaviors and the profiles of the related nodes will be tagged. Just to demonstrate a typical example, when Bob detects that Alice is a dishonest node, Alice's profile is tagged with the statement "Bob accused Alice of doing misbehavior" and Bob's profile appears the statement "Alice is a bad guy". Notice that in social networks, no one would like to be tagged as dishonest. Thus dishonest nodes never do any misbehavior which will jeopardize their reputation or affect their profiles like tagging. Moreover, dishonest nodes are also selfish in the sense each of them prefers to take care about its own reputation to covering up its suspected accomplices. As such dishonest nodes addressed here are rather restricted than Byzantine nodes [120]. Byzantine

nodes may do anything they wish. When messages reach to Byzantine nodes, they can drop or do not forward these messages to their neighbors even if requested.

To tolerate the existence of dishonest nodes with a limited vote corruption, we assume each node has at least one honest friend but it does not know exactly which friend is honest or not.

Moreover, we do not take into account the Sybil attacks since they can be solved by other work such as SybilGuard [188], and SybilLimit [187]. Without Sybil identities, the remaining attacks of dishonest nodes to be considered are wrongfully blaming. That includes the following cases: a single dishonest node blames a group of other participants, and a coalition of dishonest nodes blames another set of nodes. Dishonest nodes also attempt to spam the system with a large number of blames. Nonetheless, these attacks will affect their reputation when honest nodes execute verification procedures and tag profiles of the dishonest friends.

For example, in Fig. 3.1, node s broadcasts its values to nodes n and v. Fig. 3.1a depicts the case where s is dishonest but v is not, and in Fig. 3.1b, s is honest and v is dishonest. In both cases, n receives two different values orginated from s, one directly from s $(+5)$ and one from v (-5). As none of the messages were cryptographically secured, n cannot identify whether s sent two different values or v modified s's value. Even when n requests both nodes to send the values they sent/received, this problem could not be solved as dishonest nodes can tamper with the content of response. For instance, in Fig. 3.1a, if n requests s the value it sent to other nodes, then s replies that s sent v a value of $+5$ (to advocate the same value as the one it already sent to n), in this case, s wrongfully accuses v of being a dishonest node. And in Fig. 3.1b, if n requests v to give the value it received from s, v can reply that s sent it a value of -5 to justify its forwarded value. In this case, the problem of finding out who is dishonest cannot be solved. The only conclusion that n can draw here is that either s or v is dishonest. (Noted that we do not consider the case where both s and v are dishonest nodes because we assume that all dishonest nodes unify the opinion and give the same corrupted values.) Thus, in this case, n will tag the profiles of both s and v. It means the reputation, including the one of dishonest nodes, are finally affected. From the viewpoint of dishonest nodes, that is their unexpected circumstance because they really want to participate the polling process and bias the result without involving their reputation.

More generally, if s is not a direct friend of n: when n receives different values of the source s, it cannot distinguish whether s sent different values or the intermediate dishonest nodes modified s's value. In that case, n tags the profiles of its friends who sent it those different values. These friends also tag the profiles of their neighbors who forwarded the s's value to them, and so on. With this process, the dishonest nodes will be eventually tagged (even if some other honest nodes are also tagged) by certain honest nodes who are dishonest nodes' friends. Again, that is not a desired situation for dishonest nodes.

Actually, in a system where majority of nodes are honest, the probability to expose dishonest nodes that wrongfully accuse honest ones is high. Several tools or systems could distinguish legitimate and wrongful accusations. For instance, reputation systems like EigenTrust [114] and PowerTrust [193], spam mitigation systems like Ostra [138] and SocialFilter [169], and recommendation systems like SumUp [177] and Digg [2]. Hence, for the sake of simplicity, we do not consider the wrongful blame attack in this thesis.

We represent $\mathcal{H}(X)$ and $\mathcal{D}(X)$ as the set of honest nodes and dishonest nodes respectively in a graph $X \subseteq G$.

Paths and shortest paths

Definition 3.5 (Path)**.** *A* path p *of length* $l \in \mathbb{N}$ *is an ordered sequence of* $l+1$ *nodes such that there exists an edge connecting two consecutive nodes in the sequence:* $p = \langle u_1, u_2, \ldots, u_{l+1} \rangle$ *with*

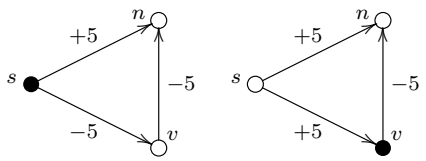


Figure 3.1: Node n cannot identify whether s or v is dishonest.

$u_i \in V$, $(u_i, u_{i+1}) \in E$, $1 \leq i \leq l$. We write $|p|$ to refer the length of path p, i.e., number of the edges of p. If $e(u_i, u_{i+1}) = 1$ then $|\langle u_i, u_{i+1}\rangle| = 1$. If a path p contains only one node, $|p| = 0$.

For two nodes $u, v \in V$, let $p(u, v)$ be a path connecting nodes u and v and $Pa(u, v)$ be the set of all such paths. We write $x \in p(u, v)$ if a path $p(u, v)$ contains a node x. For two paths $p(u_1, v_1)$, $p(u_2, v_2)$, we define the intersection of them as follows: $p(u_1, v_1) \cap p(u_2, v_2) = \{x \in V \mid x \in p(u_1, v_1) \text{ and } x \in p(u_2, v_2)\}$.

Definition 3.6 (Shortest path)**.** *The* shortest path *between two nodes u and v, denoted by $p_S(u, v)$, is the path having shortest length comparing to other paths of the set $Pa(u, v)$.*

We denote $\delta(u, v)$ as the distance, i.e., the length of the shortest path, between nodes u and v. We also illustrate by $Pa_S(u, v)$ the set of all shortest paths between two nodes u and v.

Distances, eccentricity, radius and diameter

Definition 3.7 (Ecentricity)**.** *The* eccentricity $Rad(v, G)$ *of a node $v \in V$ denotes the distance from v to the vertex farthest away from it in G: $Rad(v, G) = \max_{w \in V}\{\delta(v, w)\}$.*

Definition 3.8 (Diameter and radius)**.** *The* diameter $Diam(G)$ *of the graph G is defined as the maximal distance between any two nodes in it: $Diam(G) = \max_{u,v \in V}\{\delta(u, v)\} = \max_{v \in V}\{Rad(v, G)\}$. We illustrate the* radius *of the network by $Rad(G) = \min_{v \in V}\{Rad(v, G)\}$.*

For the single-node graph $G = (\{v\}, \emptyset)$ we suppose that $Rad(G) = Diam(G) = 1$. To our best knowledge, the distributed algorithms for computing the exact eccentricity, radius and diameter of the network take time $\mathcal{O}(N)$ [98, 153]. We present these algorithms in Section 3.3. Moreover, when G is clear from the context, we sometimes omit (G) and write the notations in the forms of abbreviations without that script such as $Rad(v)$, Rad, and $Diam$.

Neighborhoods

We rewrite the set of direct neighbors of node n as the form $\Gamma_n^1 = \Gamma(n)$ and define recursively the collection of neighbors at distance $j > 1$ from n as follows:

$$\Gamma_n^j = \{u \mid \delta(u, n) = j\} = \{u \mid u \in \Gamma_v \text{ where } v \in \Gamma_n^{j-1} \text{ and } u \notin \bigcup_{k<j} \Gamma_n^k\}$$

Honest graph

Since each node is either honest or dishonest, for the transmission between two nodes u and v, it is important to care of the honesty property of each node (i.e., checking whether node is honest or dishonest) in the paths connecting them. Particularly, if u and v are directly connected, i.e., $e(u, v) = 1$, we should investigate the honesty property of u and v. The transmission is secure

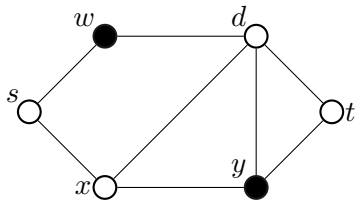


Figure 3.2: Example of social graph.

only if they are all honest and is unsecured otherwise. If $e(u,v) = 0$, we should examine all paths between u and v. For a path $p(u,v) = \langle u \equiv u_1, u_2, ..., u_m \equiv v\rangle$ (where $e(u_i, u_{i+1}) = 1$, $1 \leq i \leq m-1$), we have to check honesty property of each intermediate node u_i. The transmission in that path is secure only if all nodes are honest and we call it *honest path*. If there exists at least one honest path between u and v, it guarantees the correct information from u (or v) will approach to v (or u).

We describe, more formally, the way to check the secure transmission between nodes u and v by using concept *trust level*. Let us firstly define the value q for two directly connected nodes u, v as $q(u,v) = 1$ if u, v are honest, and $q(u,v) = 0$ otherwise. Note that the value of $q(u,u)$ depends on whether u is honest or not. The trust level for a specific path p is:

$$\varphi(p) = \begin{cases} q(u,u) & \text{if } p = \langle u\rangle \\ q(u_1,u_2).\varphi(\langle u_2,...,u_m\rangle) & \text{if } p = \langle u_1,u_2,...,u_m\rangle \end{cases} \tag{3.1}$$

Here, notation "." is a multiplication operation.

Given two nodes u and v, the trust level $\varphi(u,v)$ is given as:

$$\varphi(u,v) = \sum_{p \in Pa(u,v)} \varphi(p) \tag{3.2}$$

Obviously, $\varphi(u,v) = 0$ if u or v is dishonest. Therefore, we often use $\varphi(u,v)$ when u, v are honest. In case that transmission in path $p = \langle u_1, u_2, ..., u_m\rangle$ is secure, i.e., $\varphi(p) > 0$, then p is a honest path. For instance, consider the graph given in Figure 3.2, where we consider dishonest nodes are w and y and the remaining are honest ones. According to Formula 3.2: (i) $\varphi(w,w) = 0$ since $q(w,w) = 0$, and (ii) $\varphi(s,d) = \varphi(p(\langle s,w,d\rangle)) + \varphi(p(\langle s,x,d\rangle)) + \varphi(p(\langle s,x,y,d\rangle)) + \varphi(p(\langle s,x,y,t,d\rangle)) = 1$ since $q(s,w) = 0$ and $q(x,y) = 0$.

Definition 3.9 (Honest graph)**.** *For a graph G, there exists, for all pairs of honest nodes u, v, at least one honest path between them, then G is called* honest graph. *Formally, G is a honest graph if $\forall u, v \in \mathcal{H}(G) : \varphi(u,v) > 0$.*

3.2.2 Algorithm performance

In the centralized algorithms, the factors of performance evaluation are often the time and space complexities. But the corresponding performance consideration in the distributed system are more sophisticated. They include the assessment for the *time*, *space* and additional quantity, *message* (or *communication*). We describe here the formal definitions of these complexities which are given in [152].

Time complexity

The time complexity is used for evaluating the deadline of the algorithm completion, i.e., the time by which a node may expect to get the output of their computation. This complexity measurement for a centralized algorithm is the number of steps that program takes from the beginning to the end. Since the computation and execution of the program are occurred at one site, the measurement is robust even if that program might be paused or resumed several times. In contrast, with the presence of multiprocessors, the execution of a distributed program may take place at many sites and progress at different times. This leads to the situation where a node waits for information computed at another and the delays might be important. We consider this complexity measure in two types of networks: synchronous and asynchronous. Remember that in the synchronous system, the entire system is driven by a global clock whose pulses satisfies the following property: If a processor n sends a message to its neighbor v at pulse p then it must arrive at v before pulse $p + 1$ of v. As for the asynchronous network, the time for delivering a message from the source to the destination is not bounded, and there is no global clock to drive the actions of entire network. To overcome this difficulty, we assume that in asynchronous network, each message incurs a delay of at most one time unit.

Definition 3.10 (Synchronous time complexity)**.** *Given a network G and a synchronous distributed algorithm $\mathcal{A}$. The* time complexity *of $\mathcal{A}$ when deployed on G is the number of pulses generated during the execution of $\mathcal{A}$ on G in the worst case counting from the first node in G began the execution until the last node has terminated.*

Sometimes, in this thesis, we are interested in considering the time complexity as the maximum time complexity defined in Definition 3.11.

Definition 3.11. *Given a network G and a synchronous distributed algorithm $\mathcal{A}$. The* time complexity *of $\mathcal{A}$ when deployed on G is the maximum number of pulses generated during the execution of $\mathcal{A}$ on G in the worst case at all nodes.*

Definition 3.12 (Asynchronous time complexity)**.** *Given a network G and an asynchronous distributed algorithm $\mathcal{A}$. Under the assumption that each message transmission incurs a delay of at most one time unit, the* time complexity *of $\mathcal{A}$ when deployed on G is the maximum number of time units from the start of the execution of $\mathcal{A}$ on G to its completion in the worst case at all nodes.*

It is also noted that the assumption in the Definition 3.12 does not refer to the existence of the bounded delay in the asynchronous systems or does not restrict the prospective scenarios in any meaningful way. That assumption is used for functioning assessment. In fact, we can transform any certain scenario where the message transmission takes longer than one time unit into other one which time of transmission is between 0 and 1 without affecting to the execution of the algorithm. This may be done by dividing all current transmission times by the suitable factor, e.g., the maximum transmission time in the system.

Space complexity

The space complexity refers to the maximum memory required for the local computation of the algorithm at each node, or to the total memory used by algorithm. More formally:

Definition 3.13 (Space complexity)**.** *Given a network G and an algorithm $\mathcal{A}$. The* space complexity *(resp.* total space complexity*) of $\mathcal{A}$ when deployed on G is the maximum (resp. total) number of memory bits used by $\mathcal{A}$ at at all nodes of G in the worst case.*

Message complexity

The message complexity is the major difference between distributed systems and centralized systems. We distinguish this complexity measure with the time complexity in the centralized settings as follows: the time complexity in the centralized networks includes (i) the estimation of the deadline for the program completion, and (ii) the cost, i.e., the expected amount of computation required for the computation. While in the distributed networks, the first evaluation (i) could still be reached by the time complexity notion, but evaluation (ii) could not be applied since nodes could collaborate together to speed up the computation. Instead, the communication cost is assessed based on the message complexity in the distributed systems. This complexity takes a significant role in the decentralized networks and it relates to the amounts of information each node or entire system transmits.

Definition 3.14 (Message complexity)**.** *Given a network G and a distributed algorithm $\mathcal{A}$. Under the assumption that the* message transmission cost *over a link is 1, the* message complexity *of $\mathcal{A}$ when deployed on G is the total number of messages transmitted during the execution of $\mathcal{A}$ on G in the worst case.*

We also examine the message complexity as the number of messages that a node has sent. We define this consideration in Definition 3.15.

Definition 3.15. *Given a network G and a distributed algorithm $\mathcal{A}$. Under the assumption that the* message transmission cost *over a link is 1, the* message complexity *of $\mathcal{A}$ when deployed on G is the total number of messages a node has sent during the execution of $\mathcal{A}$ on G in the worst case.*

3.3 Preprocessing algorithms

In Section 3.1.2, we already mentioned that the knowledge about the system at each node is limited. In some networks, e.g., anonymous networks, each node does not know anything about the network topology. But in other systems, to simplify and facilitate some processing and computations, some additional information on the surrounding or partial topological knowledge may be available to each node. Moreover, some models do not supply much information about topology explicitly, instead, they implicitly give it by putting various network aspects. For example, a spanning tree or a Breadth-First-Search (BFS) tree may be known to all nodes by the following meaning: each node does not know entire the tree structure, and could not manipulate directly to some of its characteristics. On the contrary, it may recognize which of its neighbors appear as the neighbors in the tree. Namely, a node v has d links $e_1, ...e_d$ will store an array $b = \{b_1, ...b_d\}$ of boolean values where $b_i = 1$ $(i \leq d)$ indicates the link e_i is inside the tree, otherwise $b_i = 0$. Each node also identifies the neighbors in the graph that are its *parent* and *children* in the tree respectively by using a variable $parent(v)$ and $child(v)$ pointing to corresponding neighbors.

This section aims to give some procedures to compute some partial information of the network such as tree constructions, the distance between two nodes, the network diameter. These algorithms will be used as preprocessing procedures in our thesis.

3.3.1 Tree building

We here sketch some basic processes for building bread-first-search (BFS) trees, depth-first-search (DFS) trees in a distributed way given by Peleg (cf. [152]).

BFS tree

Definition 3.16 (BFS tree)**.** *BFS is a type of searching process originating from a given root where one traverses all nodes in such a way that new nodes at minimum distance to the root are discovered first, before examining the others at higher distance. The tree constructed from that process is called* BFS tree*.*

Let v_0 be the root of the BFS tree $BFS(v_0)$. Each node maintains a variable $d(v)$ indicating the distance to the root. The algorithm constructing $BFS(v_0)$ contains the following steps.

1. Each node initializes value of l: $l(v_0) \leftarrow 0$ and $l(v) \leftarrow \infty$ for $v \neq v_0$.
2. The root broadcasts the message $msg(0)$ to all its neighbors.
3. For each node v, upon receiving a message $msg(d)$ from a neighbor t, it checks whether $d+1 < l(v)$. If that is true, then v considers w as its parent in the tree, updates value $l(v) \leftarrow d+1$ and finally forwards messages $msg(d+1)$ to other friends.

In a synchronous model, each nodes updates $l(v)$ for a finite value only once. Thus it only broadcasts messages to neighbors once. Therefore, the message complexity is $\mathcal{O}(|E|)$. The runtime of the protocol is equal to the time for discovering the farthest node from the root. In the worst case scenario, it takes $\mathcal{O}(Diam)$ if that farthest node is at distance $Diam$ from the root. In an asynchronous model, under the assumption that each message transmission takes one time unit, we see that a node at distance d from v_0 has received a message $msg(d-1)$ from a certain neighbor after d time units. Thus, similar to the synchronous case, the time complexity is $\mathcal{O}(Diam)$. For the message complexity: a node v changes the value $l(v)$ at most $N-1$ times since the maximum depth of a node in the tree is $N-1$, thus it sends at most $(N-1)d_v$ messages. It infers the total messages sent by nodes in the system is $\mathcal{O}(N\sum_v d_v) = \mathcal{O}(N|E|)$.

DFS tree

Definition 3.17 (DFS tree)**.** *DFS is a type of searching process originating from a given root where one traverses all nodes in such a way the search pattern always visits new nodes as far from the root as possible, before back tracking to a certain region. Accordingly, the tree generating from that procedure is called DFS tree.*

Let us describe a distributed algorithm for building DFS tree $DFS(v_0)$ of root v_0 in the following steps.

1. The roots starts the procedure by holding a token τ.
2. For each node v:

   Upon receiving a token τ from a neighbor w **do**:

   - If $v \neq v_0$ is visited for the first time then it:
     - Sets the node w as its parent and marks w as visited neighbor.
     - Sends messages $I(v)$ to all other neighbors to inform them it is already visited.
     - Waits for all neighbors' acknowledgements.
   - If v has a neighbor u that was not yet visited, v sends the token τ to u and marks u as a visited neighbor. Otherwise v sends the token to its parent. If there is

no such node, i.e., $v = v_0$ and all neighbors have been visited, the construction terminates.

Upon receiving a message $I(w)$ from a neighbor w **do**: mark w as a visited node and send acknowledgement back to w.

The message $I(v)$ and the acknowledgement help each node to know exactly which of its neighbors were already visited and which ones were not. Since each node v transmits at most one message over an edge (including one for token, or one $I(v)$, or one acknowledgement) and each edge might not be considered by both incident nodes, the message complexity is $\mathcal{O}(|E|)$. The runtime of the algorithm includes the time for the first visiting to nodes ($\mathcal{O}(1)$ for each node) and the time for traversing DFS tree edges. Each tree edge has been visited at most twice. With N nodes, the DFS tree has $N-1$ edges; thus, the time complexity is $\mathcal{O}(N)$.

3.3.2 All-pairs of shortest paths and network diameter

In this part, we present an exact algorithm proposed by Holzer and Wattenhofer [98] for computing all pairs of shortest paths (APSP) and a by-product, the diameter, of a network in a distributed way with time $\mathcal{O}(N)$. It is noted that there is a similar algorithm to compute APSP and network diameter independently introduced by Peleg *et al.* in [153].

Communication model

We consider a synchronous system modeled as an undirected unweighted connected graph $G = (V, E)$ as presented in Section 3.2.1. Each node has a unique identifier and has no knowledge of topology except its direct neighbors. The communication model examined here is $\mathcal{CONGEST}$ [152] where nodes can send messages of size $\mathcal{O}(\log N)$ bits to its neighbors and also receive messages from them in a round. The local computation is free as it is dominated by the communication cost. In these algorithms, the time complexity, the number of rounds that the algorithms takes from the beginning until the end of the algorithm, is evaluated.

Algorithm. The algorithm is executed simultaneously by every node with the following steps.

1. An initiator node v_0 in the network builds a bread-first-search BFS tree T_0.
2. That initiator sends a token τ to traverse that tree T_0.
3. **Upon** receiving the token τ for the first time, each node $v \neq v_0$ will:
   - wait one time slot (in order to avoid congestion).
   - start constructing a BFS tree $BFS(v)$ from itself.
4. **Upon** the process $BFS(u)$ visits v for the first time, node v knows its depth in that tree, i.e., $\delta(u, v)$, and finally knows the value $d(v) = \max_w\{\delta(w, v)\}$.
5. Each node broadcasts $d(v)$ to all nodes in the tree $BFS(v)$. Then it can obtain the diameter by computing $Diam = \max_w\{d(w)\}$

Correctness. We see that both $BFS(u)$ and $BFS(w)$ could not be processed at the same time at any node v. Indeed, w.l.o.g., assume node u and w respectively start building $BFS(u)$ and $BFS(w)$ at time t_u and t_w where $t_u < t_w$. This infers, node u receives the token τ before w does, i.e., the token visits u then takes some time Δt to reach w. Since a message passing one

edge takes one time unit, $\Delta t \geq \delta(u, w)$. Moreover, a node w needs waiting one time unit before starting its $BFS(w)$, the time for establishing that procedure is $t_w = t_u + \Delta t + 1 > t_u + \delta(u, w)$. Therefore, $t_w + \delta(w, v) > t_u + \delta(u, w) + \delta(w, v) \geq t_u + \delta(u, v)$. In other words, the progress of $BFS(w)$ takes place after the one of $BFS(u)$ does at node v.

Time complexity. All BFS tree processes finish after $\mathcal{O}(Diam)$ rounds. The time the toke τ needs to traverse tree T_0 is $\mathcal{O}(N)$. The time complexity of the algorithm includes the time of building T_0 (in $\mathcal{O}(Diam)$), traversing the tree T_0 by the token τ (in $\mathcal{O}(N)$), and building the last BFS which τ established (in $\mathcal{O}(Diam)$). In conclusion, it is $\mathcal{O}(N)$.

3.4 Summary and Discussion

In this chapter, we presented a general issues including communication model, user knowledge, timing and synchrony in the distributed system. We also described the social network model that will be used throughout this thesis. The complexity measurements in the distributed system are also discussed. Moreover, we mentioned some recent decentralized preprocessing protocols for computing lengths of shortest paths, network diameter, BFS/DFS trees which are used for other algorithms later on.

In the following chapter, we will propose our first contribution for the polling problem. More precisely, we will give the design of a decentralized polling protocol not requiring any central authority or cryptography system. This protocol is deployed on the original social network and operates in a synchronous model.

Chapter 4

Synchronous Model-based Polling Protocol

Contents

4.1 Polling model 45
4.1.1 User behaviors 46
4.1.2 Social graph model 46
4.2 Polling protocol 47
4.3 Correctness 51
4.3.1 Properties of protocol 51
4.3.2 Protocol and graph without dishonest nodes 52
4.3.3 Protocol and graph with dishonest nodes 54
4.3.4 Particular networks 61
4.4 Experimental evaluation 63
4.5 Summary and discussion 65

As motivated in Chapter 1, one of the main issues in this thesis is to design distributed polling protocols to be deployed in social networks. This chapter (and next two chapters) will address to this issue by proposing a simple decentralized polling protocol that relies on the current state of social graphs.

This chapter is organized as follows. Section 4.1 describes our polling model, and introduces a family of social graphs. Section 4.2 presents our polling protocol with its correctness properties. We establish formally the relation between the protocol and the family of social graphs, and analyze different complexities to perform the polling in two cases, with and without the presence of dishonest nodes, in Section 4.3.2 and Section 4.3.3 respectively. Section 4.4 illustrates our experimental results. We conclude the chapter in Section 4.5.

4.1 Polling model

This section defines the user behaviors and presents the graph models to describe social networks. It should be noted that we consider the same assumptions given in [82, 83].

4.1.1 User behaviors

The polling problem consists of a system with N uniquely identified nodes representing users of a social network. Each participant n expresses its opinion by giving a vote $v_n \in \{-1, 1\}$. After collecting the votes of all nodes, the expected outcome is $\sum_n v_n$.

We consider here the following assumptions (more details are introduced in Chapter 1 and Section 3.2.1 of Chapter 3):

1. We consider the *synchronous* model and the network contains no crash and message loss.
2. The message transmission to all neighbors of a node takes one time unit.
3. The network includes honest and dishonest nodes.
4. Each node has at least one honest friend but it does not know exactly which friend is honest or not.
5. Each node has the global knowledge (that are the shortest path lengths between two arbitrary nodes).
6. The dishonest nodes want to misbehave to achieve these goals without affecting their reputation and to be tagged in their profiles: (i) bias the result of the election by promoting their votes or changing the values they received from other honest nodes; (ii) infer the opinions of other nodes.
7. In order to unify the opinions and not give compensating effects, all dishonest nodes make the single coalition $\mathcal{D}$ of size D and give the same corrupted values.
8. The dishonest nodes are rather restricted than Byzantine nodes.
9. We do not take into account the Sybil attacks, spam and the situation that dishonest nodes wrongly blame honest ones.
10. All nodes have to send/receive/forward messages without delaying if they are requested.

The assumptions 7, 8 and 9 are motivated in the attacker model in Section 3.2.1 of Chapter 3.

4.1.2 Social graph model

We present the social network in our problem in the form of models of social graphs as introduced in Section 3.2.1 of Chapter 3.

Recall that each node n maintains a set of direct neighbors $\Gamma(n)$ (or Γ_n) of size d_n. In addition, it also holds other two subsets of $\Gamma(n)$: a set $\mathcal{S}_n$ of *consumers* containing nodes that n sends messages to, and $\mathcal{R}_n$ of *producers* relating to nodes for which n acts as a consumer. These subsets might not be disjoint, i.e., $\mathcal{S}_n \cap \mathcal{R}_n \neq \varnothing$, as depicted in Fig. 4.1.

Like [82, 83], we use a predefined parameter $k \in \mathbb{N}$ and $k \leq \lfloor(-1 + \sqrt{3N+1})/3\rfloor$ (this parameter will be detailed in section 4.2) to present the features of our social graphs. Let $G = (V, E)$ be a social graph with the following properties:

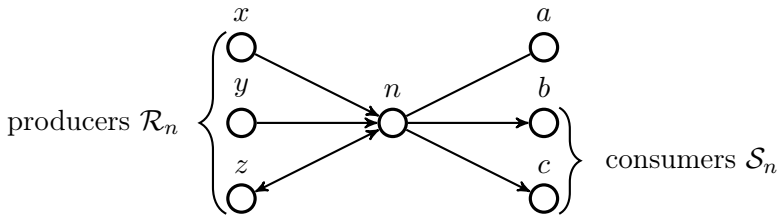


Figure 4.1: Producers and consumers of node n.

Property 4.1 (P_{g_1}). *$d_n \geq 2k+1$ and $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k+1$, for every $n \in V$.*

Property 4.2 (P_{g_2}). *G is an honest graph, i.e., for every honest nodes u, v, there exists a path $p(u, v)$ containing only intermediate honest nodes.*

Property 4.3 (P_{g_3}). *$D \leq N/10$.*

According to Property P_{g_1}, a set of consumers and a set of producers of one node have the same size and may be not disjoint. Property P_{g_2} ensures each honest node always obtains one correct version of data from other honest ones. Property P_{g_3} enables us to limit the control of dishonest users in the whole system.

From these properties, we characterize two families of graphs:

(i) $\mathcal{G}_1 = \{G \mid \mathcal{D}(G) = \varnothing$ and G satisfies $P_{g_1}\}$.

(ii) $\mathcal{G}_2 = \{G \mid \mathcal{D}(G) \neq \varnothing$ and G satisfies P_{g_1}, P_{g_2} and $P_{g_3}\}$.

Graphs in $\mathcal{G}_1$ contain only honest nodes and satisfy property P_{g_1}. Graphs in $\mathcal{G}_2$ contain honest and dishonest nodes and satisfy properties P_{g_1}, P_{g_2} and P_{g_3}.

4.2 Polling protocol

Generally, the polling protocol includes three phases (see Algorithm 1): (*i*) *Sharing*, (*ii*) *Broadcasting* and (*iii*) *Aggregating*. Phase *Sharing* describes the generation, distribution of a set of shares of each node to its neighbors as well as collecting these shares from its neighbors. In the *Broadcasting* phase, each node broadcasts messages containing the total shares, which are collected in the Sharing phase, to its direct and indirect neighbors. The last phase, *Aggregating*, shows the process that each node decides data received from other nodes and computes the final outcome.

Sharing. In this phase, each node n contributes its opinion by sending a set of shares expressing its vote $v_n \in \{-1, 1\}$ to its consumers. We inspired the sharing scheme proposed in [61] to generate shares. Namely, n generates $2k+1$ shares $\mathcal{M}_n = \{m_1, m_2, ..., m_{2k+1}\}$ where $m_i \in \{-1, 1\}$, $i = 1, 2, ..., 2k+1$ including: $k+1$ shares of value v_n, and k shares of opposite v_n's value. The intuition of this creation is to regenerate the vote v_n when the shares are summed. Later it randomly generates a permutation μ_n of $\mathcal{M}_n$, and sends shares to $2k+1$ consumers. Lines 4–9 in Algorithm 1 describe this activity. Node also receives exactly $2k+1$ shares from its producers. Note that $\mathcal{S}_n$ and $\mathcal{R}_n$ might not be disjoint.

After each node collects $2k+1$ shares from its neighbors, and sums them into *collected data* c_n (lines 10–11 in Algorithm 1), this phase completes. Fig. 4.2 illustrates an example of the

Algorithm 1: SYNCHRONOUS POLLING ALGORITHM AT NODE $n \in \{0, 1, ..., N-1\}$

Input:
v_n: A vote of node n, value in $\{-1, 1\}$
k : privacy parameter

Variables:
c_n: collected data, $c_n = 0$
$\mathcal{C}_n$: set of possible collected data
$\mathcal{C}_n[\{0, 1, ..., N-1\} \to \varnothing]$
h_n: set of final deciding collected data
$h_n[\{0, 1, ..., N-1\} \to \perp]$
$\mathcal{T}_n$: routing table
$\mathcal{T}_n[\{0, 1, ..., N-1\} \to \varnothing]$
$\mathcal{Z}_n$: sending data
$\mathcal{Z}_n[\{0, 1, ..., N-1\} \to \varnothing]$

Output: result

Main Algorithm

1 `Sharing`(v_n)
2 `Broadcasting`(n, c_n, 1, $\Gamma(n)$)
3 `Aggregating`()

Procedure `Sharing`(v_n)

4 $\mathcal{M}_n \leftarrow \{v_n\}$
5 **for** $i \leftarrow 1$ **to** k **do**
6 $\quad \mathcal{M}_n \leftarrow \mathcal{M}_n \cup \{v_n\} \cup \{-v_n\}$
7 $\mu_n \leftarrow_{rand} \mathcal{M}_n$
8 **for** $i \leftarrow 0$ **to** $2k$ **do**
9 $\quad$ send (SHARE, $\mu_n[i]$) to $\mathcal{S}_n[i]$

Upon event (receiving (SHARE, p) from neighbor r) **do**

10 **if** ($r \in \mathcal{R}_n$ **and** $p \in \{-1, 1\}$) **then**
11 $\quad c_n \leftarrow c_n + p$

Procedure `Broadcasting`(n, c_n, l_n, $\mathcal{A}_n$)

12 **foreach** ($r \in \mathcal{A}_n$) **do**
13 $\quad$ send (DATA, n, c_n, l_n) to r

Upon event (receiving (DATA, s, c_s, l_s) from t) **do**

14 **if** ($s = n$ **or** $l_s > \delta_L(s, n)$) **then** exit
15 **if** ($c_s \notin \mathcal{C}_n[s]$) **then**
16 $\quad \nu_s \leftarrow c_s$
17 $\quad \mathcal{C}_n[s] \leftarrow \mathcal{C}_n[s] \cup \{c_s\}$
18 $\quad$ Broadcast(s, ν_s, $l_s + 1$, $\Gamma(n) \setminus \{t\}$)
else
19 $\quad \nu_s \leftarrow \perp$
20 $\mathcal{T}_n[s] \leftarrow \mathcal{T}_n[s] \cup \{(t, c_s, \nu_s, l_s)\}$

Procedure `Aggregating`()

21 result $\leftarrow c_n$
22 **for** $s \leftarrow 0$ **to** $N-1$ **do**
23 $\quad$ **if** ($s \neq n$) **then**
24 $\quad\quad h_n[s] \leftarrow$ `CheckInconsistency`(s)
25 $\quad\quad$ result $\leftarrow$ result $+ h_n[s]$

Procedure `CheckInconsistency`(s)

26 **if** ($|\mathcal{C}_n[s]| = 1$) **then**
27 $\quad$ **return** $\mathcal{C}_n[s][0]$
else
28 $\quad$ **return** *correct value after verifying* $\mathcal{T}[s]$ *of neighbors*

protocol for $k = 1$. Fig. 4.2a presents desired vote of each node, whereas Fig. 4.2b depicts the sharing phase at node A. Node A would like to vote $+1$, thus, it generates a set of $2k + 1 = 3$ shares $\{+1, -1, +1\}$ which total equals to $v_A = 1$. Fig. 4.2c shows node A collects the shares from its producers and computes the collected data $c_A = 3$.

Broadcasting. In this phase, each node needs to disseminate its collected data to all other nodes in such a way that each other node eventually obtains that correct data. Briefly, this phase includes the following steps:

1. **For** the source s **do**: Send the collected data to all neighbors.
2. **For** node $n \neq s$ **do**:

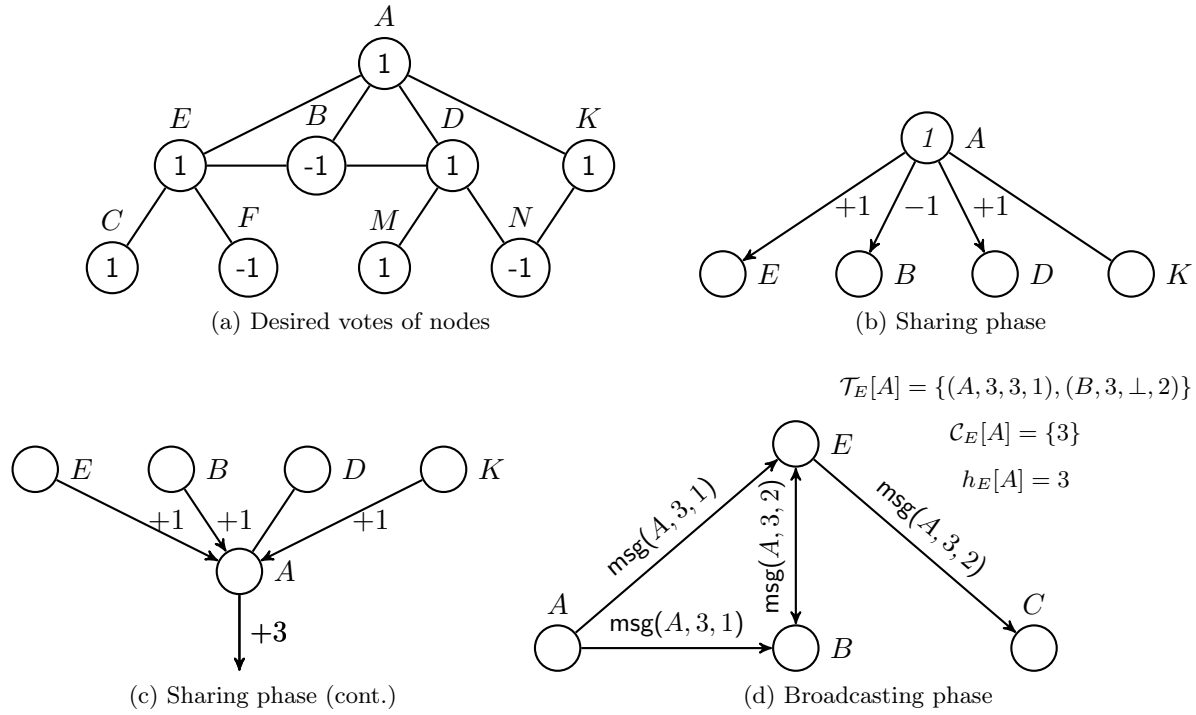
   Upon receiving the data of source s for the first time (over a neighbor t) **do**:

   2.1. Store the data in $\mathcal{C}_n[s]$.

   2.2. Forward it on every other neighbor (except t).

   Upon receiving the data again (over other neighbors) **do**: simply drop it.

More specifically, each node n encapsulates the collected data c_n with its identity n and length counter l_n, which expresses the length of the path message has passed (initially, $l_n = 1$),



(a) Desired votes of nodes

(b) Sharing phase

(c) Sharing phase (cont.)

(d) Broadcasting phase

Figure 4.2: Polling algorithm for $k = 1$.

into message msg and disseminates it to all neighbors (lines 2 and 12–13 in Algorithm 1). This action is depicted in Fig. 4.2d. When n receives from its neighbor t a message $msg(s, c_s, l_s)$ emitted from the source s, it performs the following actions (lines 14–20 in Algorithm 1):

1. *Loop detection*: n checks contents of msg and detects the loop based on the source node(line 14 in Algorithm 1). If this message is the one n has emitted earlier, i.e., $s = n$, then n simply drops the message. Otherwise, n accepts msg.

2. *Message Forwarding*: For a message passing the loop detection, n should get data c_s and forward to its friends except t.

   We see that, naively approaching, n can receive c_s from many different paths (without loop) connecting between s and n. However, the number of paths can be blown up to exponential value. More specifically, the worst case is when G is a clique and each message passes through all nodes in the network, and thus, the number of possible paths between s and n is an exponential function. This motivates us to find out an optimal solution to bound the number of messages emitted from s that n should receive without losing any necessary information.

   Instead of using naive approach, we propose other approach which is useful and more optimal: node receives messages which passed by paths with the limited length rather than accepting all. Here, for messages broadcast from s, we use bread-first expansion where the message transmission in one edge takes one time unit. Hence, we see that firstly n receives messages from s in the shortest path $p_S(s, n)$, and then from other paths of greater length. By the way, the content of messages can be changed by some intermediate dishonest nodes in the path $p \in Pa(s, n)$. Thus, we should take care the intermediate nodes. For each

intermediate node x, it receives message from s in $p_S(s,x)$ first and from the longer path later. Node n also receives message, which passed x, from the shortest path $p_S(x,n)$ first and then from other longer paths $p(x,n)$. Therefore, n receives messages, which are broadcast from s and passed x, from the paths with length $\delta(s,x)+\delta(x,n)$ first, and from other longer paths later. To take care of all possible changes in contents, n should receive all messages which already passed all intermediate nodes. And so, the maximum length of the paths passing message n should receive is $max_x\{\delta(s,x)+\delta(x,n)\}$.[7] In case that for all node x, $p_S(s,x)$ and $p_S(x,n)$ have some common nodes (different from x), n should not receive messages from the paths of length $\delta(s,x)+\delta(x,n)$ since they have a loop inside. It should receive messages from paths of length $\delta(s,n)$ instead. So, we combine all of these results, and define one value which n (resp. s) could use to determine the maximum length of paths which deliver messages from s (resp. n) to n (resp. s) as follows:

$$\delta_L(s,n) = \begin{cases} \max_{x\in U_{sn}}\{\delta(s,x)+\delta(x,n)\} & \text{if } s \neq n \wedge |U_{sn}| > 0 \\ \delta(s,n) & \text{otherwise} \end{cases} \tag{4.1}$$

where $U_{sn} = \big\{x \in V | x \neq s,n \text{ and } \exists p_1 \in Pa_S(s,x), p_2 \in Pa_S(x,n) \text{ s.t. } p_1 \cap p_2 = \{x\}\big\}$.

For a message with $l_s \in \big[\delta(s,n), \delta_L(s,n)\big]$, node n accepts and does the following activities, otherwise it simply eliminates that message (line 14 in Algorithm 1).

The activities in the case $l_s \in \big[\delta(s,n), \delta_L(s,n)\big]$ are as follows (lines 15–18 in Algorithm 1): n checks $\mathcal{C}_n[s]$, a set of possible values emitted from the source with identity s, to determine whether c_s is already presented in it:

- If c_s is not stored in $\mathcal{C}_n[s]$: n will add it into $\mathcal{C}_n[s]$, and then forward message $msg(s,\nu_s,l_s+1)$, where ν_s is value to be sent (in this case $\nu_s = c_s$), to other direct neighbors except t. All information about the messages from source s is stored in the *routing table* $\mathcal{T}_n[s]$ which is used for checking inconsistency later. This table contains the following fields: the neighbor identity from which n received message (e.g., t), the receiving value (e.g., c_s), the value to be forwarded (e.g., ν_s), and the length of the path passing message from the source s (i.e., l_s). In this case, n adds tuple (t, c_s, ν_s, l_s) into $\mathcal{T}_n[s]$.
- Otherwise: when $\mathcal{C}_n[s]$ contains value c_s, n does not need replicating that value in $\mathcal{C}_n[s]$, as well as forwarding it to other friends as it already did earlier. It just stores information in the routing table, by setting the sent value as null, i.e., $\nu_s = \perp$ (null).

Fig. 4.2d depicts the process when node E receives message emitted from A. When $msg(A,3,1)$ with length 1 arrives to E, it stores $c_A = 3$ into the set $\mathcal{C}_E[A]$ of possible collected data of source A. It then forwards $msg(A,3,2)$ with length 2 to B and C and adds a tuple $(A,3,3,1)$ into routing table of source A, i.e., $\mathcal{T}_E[A]$. Notice that A also sends $msg(A,3,1)$ to B with the same length as the one to E. Thus, B gets the same message and does the same actions like E. Node E gets forwarded message with length 2 from B. Since that is the second message having the same source and collected data, but higher length, E does not forward it. Node E just inserts one more tuple $(B,3,\perp,2)$ expressing the information received from B into routing table $\mathcal{T}_E[A]$.

Once there is no broadcasting messages in the network, this phase is over. Since each node just sends and receives a finite number of messages, and all messages eventually arrives, it is guaranteed this phase terminates correctly.

[7] See Lemma 4.1 for the correctness of this consideration.

Aggregating. In this phase, n has to decide the collected data of other nodes before calculating the final result. To make decision for node s, it checks $|\mathcal{C}_n[s]|$ (lines 26–28 in Algorithm 1): if $|\mathcal{C}_n[s]| = 1$, the single element in $\mathcal{C}_n[s]$ is chosen as a correct collected data, otherwise there exists an inconsistency and it should do the verification: requesting all routing tables $\mathcal{T}[s]$ of neighbors and indirect neighbors to check information received and forwarded by them. If one node is detected that it already sent different values of its data or its receiving information, then an alarm is raised and that node is tagged in its profile. By doing this, n also gets the correct collected data of source s.[8] So, in any case, n achieves the correct copy of collected data of source s. It then stores that value as one item $h_n[s]$ in the array h_n, which contains collected data of other nodes, and adds it into result (lines 21–25 in Algorithm 1). After checking and summing up all collected data of nodes (including its own collected data c_n), n obtains the final result (that is $\mathsf{result} = c_n + \sum_{i \neq n} h_n[i]$).
For instance, we consider Fig. 4.2d again. From formula (4.1), we see that $\delta_L(A, E) = \delta(A, B) + \delta(B, E) = 2$. After receiving message $msg(A, 3, 2)$ from node B, and updating routing table, node E makes final decision to choose value from source A. As the set $\mathcal{C}_E[A]$ is singleton, it will set $h[A] = \mathcal{C}_E[A][0] = 3$. This value will be used to compute final outcome of polling later.

4.3 Correctness

In this section, we first present the properties of our polling protocol. Second we analyze its correctness and complexities when deployed for graphs of $\mathcal{G}_1$ and $\mathcal{G}_2$. Finally we show those properties are necessary and sufficient conditions to ensure the correctness of our protocol in graphs without and with the presence of dishonest nodes.

4.3.1 Properties of protocol

In Section 4.1.1, we already introduced the characteristics of the polling model. It implies that our protocol should have some properties such that the system can run correctly with (or without) the existence of dishonest nodes. Namely, each honest node outputs the correct polling result, controls the impact from the dishonest nodes, and not disclose its private information, whereas the dishonest coalition could not control the polling process or fool an entire network without being detected. In this section, we clarify those desirable properties by stating what protocol should achieve with (or without) the existence of dishonest nodes.

Privacy. The privacy property expresses the ability of the system to prevent the private information from being leaked to the dishonest nodes. In other words, the coalition could not reveal any information of particular honest node beyond what it can deduce from its own vote, the output of computation and the shares of votes.

Definition 4.1 (Privacy)**.** *The protocol is said* private *if the dishonest nodes cannot learn anything about the vote of honest node. More formally, for any honest node n with vote v_n, there exists a negligible function $\xi(k)$ such that:*

$$Pr[\mathcal{D} \vDash v_n] \leq \xi(k) \tag{4.2}$$

where $\mathcal{D} \vDash v_n$ denotes the dishonest coalition discloses a vote v_n.

[8] See Lemma 4.6 for the detail of this verification.

Accuracy. We define the impact of dishonest nodes as the difference between the output and the expected result. In our case, vote is either "+1" or "−1", and thus, with the system of N nodes, the maximum and minimum final results are N and $-N$ respectively. This implies the maximum difference amongst the final outputs is $2N$. As defined in [80], accuracy is given by the maximum impact with respect to the maximum difference of the final outputs:

$$\sigma = \frac{1}{2N} \cdot \max_{n \in \mathcal{H}(G)} \left| \mathsf{result}_n - \sum_{i=0}^{N-1} v_i \right| \tag{4.3}$$

where result_n is the output of the poll.

Due to the absence of cryptography techniques, we here tolerate some impact from dishonest nodes on the polling output. The formal definition of accuracy property is as follows.

Definition 4.2 (Accuracy)**.** *The protocol is said* accurate *if there exists a negligible function* $\xi(k)$ *such that* $\sigma \leq \xi(k)$*.*

Termination. Termination is expressed by the characteristics that the protocol is guaranteed to eventually terminate, and the system is not in the situation of looping indefinitely, for example, node waits for message never arriving. It means the number of messages each node is supposed to send or receive in the protocol is known and finite. Noted that, as presented in our assumptions, all nodes have to send/receive messages if they are requested.

Definition 4.3 (Termination)**.** *The protocol is said* terminate *if there exist* $m_1, m_2 \in \mathbb{N}$ *such that* $|M_s| \leq m_1, |M_r| \leq m_2$*, where* M_s *and* M_r *are respectively the set of sending and receiving messages of one node.*

Definition 4.4 (Correctness of the protocol)**.** *The polling protocol is said to be* deployed correctly *on a graph with (resp. without) the presence of dishonest nodes if it preserves the privacy, accuracy, and termination (resp. accuracy and termination) properties when deployed on that graph.*

4.3.2 Protocol and graph without dishonest nodes

In this section, we consider only graphs G of family $\mathcal{G}_1$ (see Subsection 4.1.2) and analyze the correctness (including accuracy and termination) of our protocol when deployed for graphs in $\mathcal{G}_1$. Next we give spatial, message and time complexities. Finally, we show properties of $\mathcal{G}_1$ are necessary and sufficient conditions to ensure the correctness of our protocol. Here we do not consider the privacy property as there is no dishonest nodes in this case.

Correctness

Theorem 4.1 (Correctness)**.** *Consider a polling system of size* N *with only honest nodes where each node* n *expresses a vote* v_n*. The polling algorithm is guaranteed to eventually terminate and each node outputs* $\sum_{n=0}^{N-1} v_n$*.*

Proof. (Accuracy). In the sharing phase, each node n sends a set of shares $\mathcal{M}_n = \{m_{n_1}, m_{n_2}, ..., m_{n_{2k+1}}\}$ to its consumers where $\sum m_{n_i} = (k+1).v_n + k.(-v_n) = v_n$, and also receives a set of shares $\{m'_{n_1}, m'_{n_2}, ..., m'_{n_{2k+1}}\}$ from its $2k+1$ producers to obtain a collected data $c_n = \sum_{j=1}^{2k+1} m'_{n_j}$. With the assumption that there is no dishonest node and without crash or message loss, each message from the source successfully reaches the destination, and thus the set of all sending

shares of all nodes will be exactly coincided with the set of all receiving shares of all nodes, namely:

$$\bigcup_{V}\{m_{n_1}, m_{n_2}, ..., m_{n_{2k+1}}\} = \bigcup_{V}\{m'_{n_1}, m'_{n_2}, ..., m'_{n_{2k+1}}\}$$

In the broadcasting phase, each node n broadcasts its collected data to its direct neighbors, then they do honestly forward that value to neighbors of neighbors of n and so on. Each node's data is finally arrived to all other ones. It infers that the array h_n contains all collected data of all nodes in the system and these values comes from all the receiving shares of all the nodes. Consequently, the final computation gives us the value which is also our expected outcome:

$$\mathsf{result} = c_n + \sum_{\substack{0 \le i < N \\ i \ne n}} h_n[i] = \sum_{i=0}^{N-1} c_i = \sum_{i=0}^{N-1}\sum_{j=1}^{2k+1} m'_{i_j} = \sum_{i=0}^{N-1}\sum_{j=1}^{2k+1} m_{i_j} = \sum_{i=0}^{N-1} v_i$$

.

Proof. (Termination). In the sharing phase, each node receives a finite number $(2k+1)$ of messages. In the broadcasting phase, a node n also receives the finite number of messages from source s, because it just accepts messages passed through the paths of length belonging to the interval $\big[\delta(s,n), \delta_L(s,n)\big]$. Moreover, as every node sends the required number of messages and they are eventually arrived to destination, each phase completes. The algorithm has a finite number of phases, hence, by Definition 4.3, the protocol is ensured to finally terminate.

□

Asymptotic complexity

We analyze the spatial, message and time complexities of the protocol in Propositions 4.1–4.3.

Proposition 4.1 (Spatial complexity)**.** *The total space each node n must hold is $\mathcal{O}(k + Nd_n)$.*

Proof. Each node n needs to maintain the set of $2k+1$ consumers, the set of $2k+1$ producers, the list of d_n direct neighbors, the set $\mathcal{C}_n$ of possible collected data of other nodes, the routing table $\mathcal{T}_n$, the set h_n of $N-1$ final deciding collected data. Without the presence of dishonest nodes, node n just inserts into $\mathcal{C}_n[s]$ one value emitted from source s through the shortest path between s and n, i.e., $|\mathcal{C}_n[s]| = 1$, and thus $|\mathcal{C}_n| = N-1$. The worst case for the routing table $\mathcal{T}_n[s]$ is when all shortest paths between s and n are passed by its friends. In other words, $\mathcal{T}_n[s]$ contains at most d_n rows. It infers that $|\mathcal{T}_n| \le (N-1)d_n$. Therefore, the spatial complexity in this case is $\mathcal{O}(k) + \mathcal{O}(d_n) + \mathcal{O}(N-1) + \mathcal{O}((N-1)d_n) = \mathcal{O}(k + Nd_n)$. □

Proposition 4.2 (Message complexity)**.** *The number of messages sent by a node n is $\mathcal{O}(k+Nd_n)$.*

Proof. In the sharing phase, a node n sends $2k+1$ messages to its consumers. In the broadcasting phase, it sends d_n messages containing collected data to all of its direct neighbors. It also takes a role as an intermediate node by forwarding message to all of its neighbors except the node it got that message. For each collected data emitted from source s, the number of messages node n forwards is equal to the number of elements in the set $\mathcal{C}_n[s]$. The cause is that, after receiving the first message of value c_s passed by the shortest path with length $\delta(s,n)$, n stores c_s into $\mathcal{C}_n[s]$, forwards message to $(d_n - 1)$ neighbors, and never does this action for the message having the same value but higher length later. Thus the number of forwarded messages is $|\mathcal{C}_n|(d_n - 1)$. As there is no dishonest node, node n just inserts into $\mathcal{C}_n[s]$ one value emitted from source s through the shortest path between s and n, i.e., $|\mathcal{C}_n[s]| = 1$, and thus $|\mathcal{C}_n| = N-1$. Accordingly, the message complexity is $\mathcal{O}(2k+1) + \mathcal{O}((N-1)(d_n-1)) = \mathcal{O}(k + Nd_n)$.

□

Proposition 4.3 (Time complexity)**.** *The protocol operates in* $\mathcal{O}(Diam(G))$.

Proof. The sharing phase operates in one time unit as a node broadcasts shares to all its consumers at the same time. In the broadcasting phase, node n broadcasts its collected data to all other nodes. The farthest node is at distance $Diam(G)$. Thus this phase takes in $\mathcal{O}(Diam(G))$. Therefore, the time complexity is $\mathcal{O}(Diam(G))$.

□

The necessary and sufficient conditions of graph without dishonest nodes

In the following part, we examine the necessary and sufficient conditions for our protocol to be deployed correctly on graphs without the existence of dishonest nodes.

Theorem 4.2. *The properties of* $\mathcal{G}_1$ *are the necessary and sufficient conditions for the polling protocol to be deployed correctly in the system without dishonest nodes.*

Proof. The sufficient condition ($\Rightarrow$) is proved by Theorem 4.1. We only examine the remaining of this theorem, the necessary condition ($\Leftarrow$). Consider a general graph G which our polling protocol can be deployed correctly (without the presence of dishonest nodes). We will show $G \in \mathcal{G}_1$. In the sharing phase of the protocol, each node n sends (resp. receives) exactly $2k+1$ messages to (resp. from) consumers (resp. producers), i.e., $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k+1$, and these two sets $\mathcal{S}_n$ and $\mathcal{R}_n$ might not be disjoint, thus, $d_n \geq 2k+1$. Therefore, to apply protocol correctly, G must have the property P_{g_1}, and we have, $G \in \mathcal{G}_1$. □

4.3.3 Protocol and graph with dishonest nodes

In this section, we revisit the relation between protocol and graph, but approach it with the presence of D dishonest nodes. We consider graphs G of family $\mathcal{G}_2$ and analyze the correctness (including privacy, accuracy and termination) of our protocol when deployed for these graphs. Next we give spatial, message and time complexities. Finally, we show the properties of $\mathcal{G}_2$ are necessary and sufficient conditions to ensure the correctness of our protocol. Here we do not consider the termination property as it is similar to Theorem 4.1.

Privacy. In this part, we study the probability that a node vote is revealed by a dishonest coalition.

Theorem 4.3. *The probability that a coalition of* D *dishonest nodes reveals an honest node's vote when it gets* $k+1$ *identical shares from that node is bounded by* $\sum_{m=k+1}^{2k} \left(\frac{D}{N}\right)^m \left(\frac{1}{2}\right)^{2k+1-m}$.

Proof. The probability that a coalition $\mathcal{D}$ discloses a node vote v is equal to the probability of the event the coalition $\mathcal{D}$ gets at least $k+1$ shares of that node in which $k+1$ ones are identical. Let X and Y be respectively the number of shares (generating from vote v) and the number of identical shares of value v that a coalition $\mathcal{D}$ obtains. We denote $\mathcal{E}(A)$ be the event of variable A. The mentioned probability is:

$$
\begin{aligned}
Pr[\mathcal{D} \vDash v] &= Pr(\mathcal{E}(X \geq k+1).\mathcal{E}(Y = k+1)) \\
&= \sum_{m=k+1}^{2k} Pr(\mathcal{E}(X = m).\mathcal{E}(Y = k+1)) \\
&= \sum_{m=k+1}^{2k} Pr(\mathcal{E}(X = m)).Pr(\mathcal{E}(Y = k+1/X = m))
\end{aligned}
$$

It is easy to see that $Pr(\mathcal{E}(X = m)) = \binom{D}{m}/\binom{N}{m}$ where $k+1 \leq m \leq 2k$.

Moreover, $Pr(\mathcal{E}(Y = k+1/X = m))$ is the probability to get $k+1$ identical shares in the set of size m which belongs to the set of $2k+1$ elements containing $k+1$ ones of value v and k ones of value $-v$, that is, $Pr(\mathcal{E}(Y = k+1/X = m)) = \binom{k+1}{k+1}\binom{k}{m-(k+1)}/\binom{2k+1}{m} = \binom{k}{m-(k+1)}/\binom{2k+1}{m}$.

Consequently, we have

$$\begin{aligned} Pr[\mathcal{D} \vDash v] &= \sum_{m=k+1}^{2k} \left[\binom{D}{m}/\binom{N}{m}\right].\left[\binom{k}{m-(k+1)}/\binom{2k+1}{m}\right] \\ &= \sum_{m=k+1}^{2k} \frac{D(D-1)\dots(D-m+1)}{N(N-1)\dots(N-m+1)}.\frac{k(k-1)\dots(m-k)}{(2k+1)2k\dots(m+1)} \\ &< \sum_{m=k+1}^{2k} (\frac{D}{N})^m (\frac{1}{2})^{2k+1-m} \end{aligned}$$

□

Theorem 4.4. *The probability that $D < N/2$ dishonest nodes reveal an honest node's vote when they get $k+1$ identical shares from that node is bounded by $\frac{k}{2^{2k+1}}$, and the protocol is private.*

Proof. By Theorem 4.3, we have:

$$Pr[\mathcal{D} \vDash v] < \sum_{m=k+1}^{2k} (\frac{D}{N})^m (\frac{1}{2})^{2k+1-m} < \sum_{m=k+1}^{2k} (\frac{1}{2})^m (\frac{1}{2})^{2k+1-m} = \frac{k}{2^{2k+1}}$$

Function $\xi(k) = \frac{k}{2^{2k+1}}$ is negligible (since $\xi(k) \leq 1/k^4$, i.e., it decreases as $1/k^4$). By Definition 4.1, our protocol is private. □

If the poll outcome is N (resp. $-N$), it implies all nodes vote for "+1" (resp. "−1") and they all are disclosed. Moreover, w.l.o.g., assume each dishonest node always votes for "−1". Thus, if the result is $N - 2D$ (resp. $-N$) then it implies all honest nodes vote for "+1" (resp. "−1"). Without considering this case, i.e., all honest nodes do not vote for the same option, Theorem 4.5 shows us the maximum number of votes the dishonest coalition could discover.

Theorem 4.5. *If all nodes do not vote for the same option, a coalition of D dishonest nodes can reveal at most $2D$ votes of honest nodes.*

Proof. A node receives $2k+1$ shares, hence, the dishonest coalition collects at most $D(2k+1)$ shares. Moreover, a vote v of one node is revealed if the coalition obtains $k+1$ identical shares of value v. Thus the dishonest coalition recovers at most $\lfloor \frac{D(2k+1)}{k+1} \rfloor \leq 2D$ votes. □

Accuracy. We present the accuracy based on the ability of the honest nodes to get correct output and to control the impact from dishonest nodes. We first justify clearly the condition for receiving broadcast messages from neighbors of each node that we use in the broadcasting phase.

As mentioned in Algorithm 1, in the naive approach, each node broadcasts its data and also forwards the received data which is not emitted from it. Despite the use of richer social graph structures, one node can receive/send so many duplicated messages from/to other nodes. This leads to flooding the local storage. We propose other simple optimal solution: each node n should receive broadcasting messages from the paths of length $l_s \in [\delta(s,n); \delta_L(s,n)]$. We use

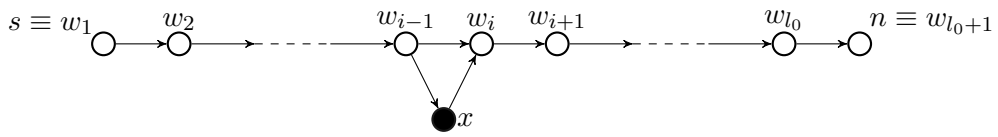


Figure 4.3: A path $\omega = \langle s \equiv w_1, w_2, ..., w_{i-1}, x, w_i, ..., w_{l_0+1} \equiv n \rangle$ of length $\delta_L(s,n) + 1$.

breadth-first expansion from s with the assumption that transmission in one edge takes one time unit. This gives the fact that node n receives the broadcasting messages from the shortest paths first, and later it receives messages from the paths which length are not greater than $\delta_L(s,n)$. Moreover, for each message generated by s and passed from the path p which length satisfies the condition $\delta(s,n) \leq |p| \leq \delta_L(s,n)$, n checks whether the collected data c_s exists in $\mathcal{C}_n[s]$ or not. If $\mathcal{C}_n[s]$ does not contain c_s, n inserts c_s into $\mathcal{C}_n[s]$ and then forwards to other neighbors except the node which just sent message to n. Otherwise, c_s is neither stored in $\mathcal{C}_n[s]$ nor forwarded. This way of creation enables to store only distinct values in $\mathcal{C}_n[s]$.

To show the correctness of our optimal approach, we have to prove that all of (distinct) information that n receives by using naive approach are stored in $\mathcal{C}_n[s]$ of the optimal approach and vice versa. More formally, let us denote:

- $N_{s,n}$ as the set of all paths between s and n in the naive approach;
- V_1 as the set of possible collected data in the naive approach;
- $O_{s,n}$ as the set of all paths between s and n in the optimal solution;
- V_2 as the set of possible collected data in the optimal solution.

We prove the correctness of the optimal approach by showing that $V_1 = V_2$ in Lemma 4.1.

Lemma 4.1. *Prove that* $V_1 = V_2$.

Proof. Let $c_n^s(p)$ be the collected data receiving from a path $p = p(s,n)$.

We have $V_1 = \{c_n^s(p) | p \in N_{s,n}\}$, and $V_2 = \{c_n^s(p) | p \in O_{s,n}\}$.

It is easy to see that $V_2 \subseteq V_1$ since $O_{s,n} \subseteq N_{s,n}$.

We now prove that $V_1 \subseteq V_2$, i.e., $\forall c \in V_1 \Rightarrow c \in V_2$, by contradiction.

Indeed, assume the contrary, i.e., $\exists \omega \in N_{s,n} : c = c_n^s(\omega) \in V_1$ and $c \notin V_2$.

Since $O_{s,n} = \{p \in Pa(s,n) | \delta(s,n) \leq |p| \leq \delta_L(s,n)\}$, it means that (i) $|\omega| < \delta(s,n)$ or (ii) $|\omega| > \delta_L(s,n)$.

Case (i) cannot occur since $\delta(s,n) = |p_S(s,n)| \leq |p|, \forall p \in Pa(s,n)$. Thus, we only consider case (ii). W.l.o.g., assume $|\omega| = l_0 + 1$ where $l_0 = \delta_L(s,n)$.

Let us consider a path $p_0(s,n) = \langle s \equiv w_1, w_2, ..., w_{l_0}, w_{l_0+1} \equiv n \rangle$ (where $w_i \neq w_j$ if $i \neq j$) of length l_0 as depicted in Fig. 4.3.

Because n gets value c but does not put it into V_2, this value must be transferred by a certain node w_i in the path p_0 and is also emitted from a certain node x which is not in the path p_0 but connects to w_i. W.l.o.g., suppose x connects to w_{i-1} and the path ω is: $\omega = \langle s \equiv w_1, w_2, ..., w_{i-1}, x, w_i, ..., w_{l_0+1} \equiv n \rangle$ of length $l_0 + 1$. (We assume this for easily understanding the proof. In general, a node x might not connect to other nodes in p_0 (except w_i) but it is contained in a certain path such as $\langle s \equiv w'_1, w'_2, ..., w'_{i-1} \rangle$. In that case, we can consider the following path: $\omega' = \langle s \equiv w'_1, w'_2, ..., w'_{i-1}, x, w_i, w_{i+1}, ..., w_{l_0+1} \equiv n \rangle$. We see that the role of ω and ω' in the proof are equivalent.)

Firstly, we prove that x cannot connect to other nodes in p_0, except w_i and w_{i-1}, i.e., $e(x, w_j) = 0$, $\forall j \notin \{i, i-1\}$. Indeed, if $\exists j$: $e(x, w_j) = 1$, $i < j \leq l_0 + 1$, then there exists a message delivered by the path $p' = \langle s \equiv w_1, w_2, ..., w_{i-1}, x, w_j, w_{j+1}, ..., w_{l_0+1} \equiv n \rangle$ of length $|p'| = i - 1 + (l_0 + 1 - (j-1)) \leq l_0$ contains value c. Moreover, as $p' \in O_{s,n}$, we have $c \in V_2$. Contradiction!

So, path $\langle s, w_2, ..., w_{i-1}, x \rangle \in Pa_S(s, x)$ and path $\langle x, w_i, w_{i+1}, ..., w_{l_0+1} \equiv n \rangle \in Pa_S(x, n)$. We have: $l_0 = \delta_L(s, n) = \max_y\{\delta(s, y) + \delta(y, n)\} \geq \delta(s, x) + \delta(x, n) = i - 1 + (l_0 + 1 - (i-1)) = l_0 + 1$. This inequation gives us the contradiction. Consequently, $V_1 \subseteq V_2$.

In conclusion, $V_1 = V_2$. □

Lemma 4.2. *In the broadcasting phase, if an honest node n broadcasts its collected data c_n then all other honest nodes will eventually receive that value.*

Proof. By Lemma 4.1, for each honest node, it obtains a set of possible collected data V_2 containing all values which could be obtained from all paths of the naive approach. This also includes the honest paths which give the correct collected data. This yields the desired result. □

Corollary 4.1. *If $|V_2| = 1$ then the single element of V_2 is a correct collected data.*

Proof. This proof is given from Lemma 4.2. □

Consider the network with privacy conscious settings. In such a network, each node has no knowledge to calculate bounds for the path lengths between it and other nodes. In that case, in the broadcasting phase, after receiving a message from source s, node n could not check the length of the path delivering that message (i.e., n could not check the condition $l_s \in \big[\delta(s, n), \delta_L(s, n)\big]$). It just verifies the set $\mathcal{C}_n[s]$ and stores (and forwards, resp.) information which has never been received (and forwarded, resp.) earlier. We call this approach "privacy-conscious one". Notice that, this approach is different from the naive one. In the naive approach, node n floods network by receiving/forwarding messages: it gets all messages, then puts c_s into $\mathcal{C}_n[s]$, and transfers it to other friends without checking whether that information is already existed in $\mathcal{C}_n[s]$.

Lemma 4.3. *By using privacy-conscious approach, no information from s to n is lost.*

Proof. Let V_3 be the set of possible collected data in the privacy-conscious approach. We first prove that $V_3 = V_2 = V_1$. Indeed, from the definition of V_1 for the naive approach above, it is easy to see that $V_3 \subset V_1$. Moreover, in the privacy-conscious approach, node n receives all messages without checking the length of the path delivering them, and thus $V_2 \subset V_3$. By Lemma 4.1, $V_1 = V_2$, hence, we have $V_3 = V_2 = V_1$. □

In the following part, we first present all capabilities of dishonest nodes to affect on poll outcome in Definition 4.5. Then we examine the impact of the dishonest nodes on accuracy.

Definition 4.5 (Dishonest capabilities)**.** *A dishonest node may affect the poll outcome with the following misbehaviors:*

1. *In the sharing phase, since a node can only generate and send shares to its consumers it is assigned (otherwise the shares are dropped) and there are at most $2k + 1$ consumers to be assigned, it must send at most $2k + 1$ shares in which at most $k + 1$ ones are identical. Hence a dishonest node may give the misbehavior by sending more than $k + 1$ (but not greater than $2k + 1$) identical shares.*

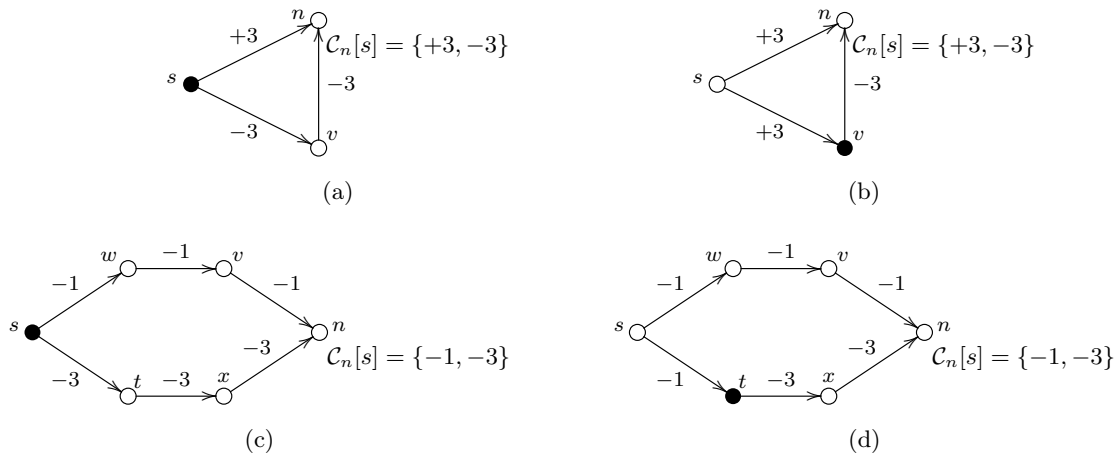


Figure 4.4: Misbehaviors of the dishonest nodes in the broadcasting phase. Node n receives different values emitted by the source s. (a) and (b) s and n connect together; (c) and (d) s and n are not direct friend of each other.

2. *It inverts each receiving "+1"-share into a "−1"-share to decrease the collected data.*

3. *In the broadcasting phase, it modifies the collected data of other honest node or sends a forged message.*

4. *It broadcasts or forwards inconsistent data.*

Lemma 4.4 (Sharing)**.** *After sending a set of shares, a dishonest node may affect at most* $2k+2$ *to the final result.*

Proof. In the sharing phase, a dishonest node may give the first misbehavior (1) presented in Definition 4.5. From the standpoint of the dishonest coalition, the best case is when it votes $v = +1$ but sends all $2k+1$ shares of value "−1". As a node is allowed to send only a set of shares summing to -1 or $+1$, this misbehavior affects the final result up to $2k + 1 + 1 = 2k + 2$. □

Lemma 4.5 (Computing collected data)**.** *After computing the collected data, a dishonest node may affect to the final result by* $4k + 2$.

Proof. Each node receives $2k + 1$ shares generating from its $2k + 1$ producers. After receiving the set of shares, a dishonest node can modify the data by inverting all shares of "+1" to shares of "−1" (misbehavior (2) in Definition 4.5). It can also create a forged data. But in any case, the dishonest node can only modify and create at most $2k + 1$ shares. The best case, from the viewpoint of the dishonest coalition, is when it gets $2k + 1$ shares of "+1" and inverts all into "−1"-shares. Thus, this attack gives the impact up to $2(2k + 1) = 4k + 2$. □

Corollary 4.2. *A dishonest node that corrupts the collected data to be out of the range* $[-2k - 1, 2k + 1]$ *is detected with certainty.*

Proof. The proof is yielded from Lemma 4.5 since a node sums $2k + 1$ shares and each share is either $+1$ or -1. □

Lemma 4.6 (Broadcasting collected data)**.** *There exists a public verification scheme that detects a dishonest node broadcasting (resp. forwarding) inconsistent copies of its (resp. other node's) collected data.*

Proof. Recall that, as presented in Sub-section 4.1.1, we do not take into account the Sybil attacks, spam, and wrongfully blaming from dishonest nodes. Thus the capabilities of dishonest nodes in the broadcasting phase are either modifying the receiving data or modifying the path length value. The latter case (i.e., modifying the path length value) could be detected due to the synchronous model. If the broadcasting phase starts at time t (for all nodes), then at time t' each node must receive messages with length $t' - t$. Notice that, the dishonest nodes cannot delay in sending or forwarding messages. Moreover, because nodes can guess the path length between any two nodes in the system, they can detect easily this kind of attack. Hence, we here concentrate only the former case, i.e., dishonest nodes modify the receiving collected data.

An example of the attack of changing collected data is depicted in Fig. 4.4a: node n receives two different values from source s, one directly from s $(+3)$ and one from v (-3). If n requests s the value it sent to other nodes, and s replies that he/she sent value $+3$ to v (to advocate the same value as the one it already sent to n), in this case, s will indirectly wrongfully accuse v as dishonest node, because according to the information from s, v has forwarded to n the different value from the one it received from s, and thus, v must be dishonest and tagged. Actually, in a system that honest nodes are majority, the probability that dishonest nodes are exposed when wrongly accusing honest ones is high. Like in the example, if v is wrongly accused by a small number of nodes, the allegation would be in doubt and not be considered, and the accuser s would be finally backfired. By the way, we do not allow this kind of blame in the system, and assume that no node, which does not wrongly blame other nodes, would like to be tagged as dishonest.

To detect that misbehavior, we suggest the following verification scheme: requesting and checking routing tables of neighbors. More particularly, when an honest node n detects inconsistency of the collected data from a source s, i.e., $\mathcal{C}_n[s] = \{w_1, w_2, \ldots, w_l\}$ where $l > 1, w_i \neq w_j, i \neq j$, for each $w_i \in \mathcal{C}_n[s]$, n checks in the routing table $\mathcal{T}_n[s]$ to find out all neighbors $\{v_{i_1}, v_{i_2}, \ldots, v_{i_j}\}$ that sent w_i to n. Then n sends all entries of $\mathcal{T}_n[s]$ which contain the values $\mathcal{C}_n[s]$, as well as requests to check the inconsistency of the value w_i to its neighbors, neighbors of neighbors, and so on. There exists some intermediate honest nodes (in the path between s and n), who are the nearest (direct or indirect) neighbors w.r.t. dishonest nodes and detect the violation.

We prove this Lemma by induction on the distance $l = \delta(s, n)$.

1. If $l = 1$: Fig. 4.4a and 4.4b show that node n receives two different values of source s, and it cannot identify whether s sent two different values or v modified s's value . As mentioned in Subsection 3.2.1 of Chapter 3, in this case, if we allowed wrongly blaming, the problem of finding out who is dishonest cannot be solved since dishonest nodes can tamper with all information. The only conclusion that n can draw is either s or v is dishonest. Thus, n will tag the profiles of both s and v as the dishonest nodes. From the viewpoint of dishonest nodes, that is their unexpected situation because they really want to participate the polling protocol and bias the polling result, but not to be accused or tagged by any nodes. However, without the wrongfully blaming attack, n can request s (resp. v) the value it sent (resp. received) to v (resp. from s), and s (resp. v) cannot spoof the information it sent (resp. received). With this method, the dishonest node will be exposed.

2. If $l > 1$: when n receives two different values of the source s (since all dishonest nodes give

the same corrupted values, n receives at most two different values which include one correct value and one corrupted value), it cannot distinguish whether s has sent inconsistent values or the intermediate dishonest nodes have modified s's value (see Fig. 4.4c and 4.4d). As discussed in Subsection 3.2.1 of Chapter 3, node n can tag its direct friends who sent those different values to it. Those friends do the same activities with their friends, and so on. Thus, dishonest nodes' profiles are finally tagged. In addition, without the wrongfully blaming attack, n can also request its friends, who have sent different values to it, to send the values they received form their friends. These neighbors have to reply to n the correct values they received. These neighbors also do the same actions as n does with their neighbors, and so on. By this process, dishonest nodes will be exposed.

□

Theorem 4.6 (Accuracy)**.** *Every dishonest node may affect the expected final result up to* $6k+4$.

Proof. By Lemmas 4.1-4.6, each dishonest node affects to the final outcome at most $2k + 2 + 4k + 2 = 6k + 4$. Thus, $\sigma = \frac{6k+4}{2N} = \frac{3k+2}{N}$.

Consider a function $\xi(k) = \frac{3k+2}{N}$. With $k \leq \lfloor(-1 + \sqrt{3N+1})/3\rfloor$ and N is fixed, then $\xi(k) \leq 1/k$. It means $\xi(k)$ decreases as $1/k$; thus, it is negligible. By Definition 4.2, the protocol is accurate. □

Asymptotic complexity We here examine the complexities of the protocol when deployed for the graphs of family $\mathcal{G}_2$ of size N with the presence of D dishonest nodes.

Proposition 4.4 (Spatial complexity)**.** *The total space each node* n *must hold is* $\mathcal{O}(k+NDd_n)$.

Proof. Each node n needs to maintain the set of $2k + 1$ consumers, the set of $2k + 1$ producers, the list of d_n direct neighbors to contact, the set $\mathcal{C}_n$ of possible collected data of other nodes, the routing table $\mathcal{T}_n$, the set h_n to store the final choosing collected data $(N - 1)$. For $\mathcal{C}_n$, in the worst case, for each honest source s, dishonest coalition can forward to n at most D different values, and n also receives one correct value from the honest path. Hence, the maximum size of $\mathcal{C}_n$ is $(N - 1)(D + 1)$. Likewise, the worst case of $\mathcal{T}_n[s]$ is when all paths between source s and n are passed by n's friends and all dishonest nodes. Since there are d_n neighbors and D honest nodes, we have the size of $\mathcal{T}_n[s]$ is at most Dd_n, and thus the maximum size of $\mathcal{T}_n$ is $(N - 1)Dd_n$. Consequently, the total space each node must hold in the worst case is $\mathcal{O}(k) + \mathcal{O}(d_n) + \mathcal{O}((N - 1)(D + 1)) + \mathcal{O}((N - 1)Dd_n) + \mathcal{O}(N - 1) = \mathcal{O}(k + NDd_n)$. □

Proposition 4.5 (Message complexity)**.** *The number of messages a node* n *sent is* $\mathcal{O}(k+NDd_n)$.

Proof. A node n sends $2k + 1$ messages to its direct neighbors in the first phase, broadcasts d_n messages in the broadcasting phase to all of its direct neighbors, and hence, totally $|\mathcal{C}_n|(d_n - 1)$ messages. The worst case for $\mathcal{C}_n$ is similar to the one considered Proposition 4.4, i.e., $|\mathcal{C}_n| = (N-1)(D+1)$, and thus, we have the message complexity is $\mathcal{O}(2k+1)+\mathcal{O}((N-1)(D+1)(d_n - 1)) = \mathcal{O}(k + NDd_n)$.

□

Proposition 4.6 (Time complexity)**.** *The protocol operates in* $\mathcal{O}(Diam(G))$.

Proof. The sharing phase operates in one time unit as a node broadcasts shares to all its consumers at the same time. In the broadcasting phase, node n sends collected data c_n to d_n neighbors which takes one time unit. The collected data is delivered to some node s through the path with length at most $\delta_L(n, s) \leq 2.Diam(G)$. Therefore, the time complexity is $\mathcal{O}(Diam(G))$.

□

The necessary and sufficient conditions of graph with dishonest nodes

We analyze the necessary and sufficient conditions for our protocol to be deployed correctly for graphs of family $\mathcal{G}_2$.

Theorem 4.7. *The properties of $\mathcal{G}_2$ are the necessary and sufficient conditions for the polling protocol to be deployed correctly in the system with the presence of dishonest nodes.*

Proof. By Theorems 4.3-4.6, we already showed the sufficient condition ($\Rightarrow$), i.e., our protocol works correctly with graphs of $\mathcal{G}_2$ and preserves its properties.

We only need to clarify the remaining proof ($\Leftarrow$) of this theorem. Assume we have a general graph G. We approach the proof by sketching step by step the requirements G should obtain so that all properties of protocol are guaranteed.

In the sharing phase of the protocol, each node n sends (or receives) exactly $2k+1$ messages to neighbors, i.e., $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k+1$, and these two sets $\mathcal{S}_n$ and $\mathcal{R}_n$ might not be disjoint, thus, $d_n \geq 2k+1$. Therefore, to apply protocol correctly, G must satisfy property P_{g_1}.

In addition, in the broadcasting phase of protocol, it can be inferred from Lemma 4.2, in order to make decision about the collected data of source s, node n has to check $|\mathcal{C}_n[s]|$. If $|\mathcal{C}_n[s]| > 1$ then there exists an inconsistency, and n starts the verification process. In that procedure, it has to request its neighbors to send the routing tables $\mathcal{T}[s]$, and its neighbors do the same activities with others. The security in this situation is the protection of the table transmission so that the honest node has some witnesses to expose the dishonest ones. We do not take into account the case that dishonest nodes blame wrongfully other honest ones in the sense they cannot modify the value that honest nodes sent to or forwarded in the content of the $\mathcal{T}$. Suppose G is not an honest graph. Then for every honest nodes u and v, every path $p(u, v)$ contains at least one dishonest nodes. And thus, the routing message always passes throughout a dishonest node, the security property of protocol is not preserved. Consequently, G must be an honest graph, or satisfies P_{g_2}.

We prove that G has property P_{g_3}. Indeed, let α, in average, be the proportion of nodes voting for "+1". The expected final outcome is $\alpha N.1 + (1-\alpha)N.(-1) = (2\alpha - 1)N$. It can be inferred from Theorem 4.6 that the maximum impact from dishonest coalition is $(6k+4)D$. The final result is not affected if $(6k+4)D \leq |2\alpha - 1|N$. Since $k \geq 1$, $0 \leq \alpha \leq 1$, we have $D \leq \frac{N|2\alpha-1|}{6k+4} \leq \frac{N}{10}$.

In conclusion, G is a member of family $\mathcal{G}_2$. □

4.3.4 Particular networks

As shown in the previous sections, our polling protocol is correct for graphs in the family $\mathcal{G}_2$. Here we illustrate some particular graphs of this family. We also prove in this part our family of graphs is more general than other families of graph structures deployed in other works.

Layered networks. The nodes are distributed into layers. Each node in one layer links to all other nodes in neighboring layer. The number of nodes in each layer might be different. Each layer has at least one honest node. Fig. 4.5 shows examples of this network in which arrows represent the direction of the transmission of shares in the first phase of the protocol (i.e., $u \to v$ means u is a producer of v and v is a consumer of u). Each node in these graphs has $2k+1 = 3$ consumers and producers. Moreover, as each honest node in layer V_i links directly to all honest nodes in V_{i+1} (and vice versa), for any two honest nodes in two arbitrary layers (even they are in the same layer), there exists an honest path connecting them. Thus, this kind of networks fulfills the property P_{g_2} of $\mathcal{G}_2$.

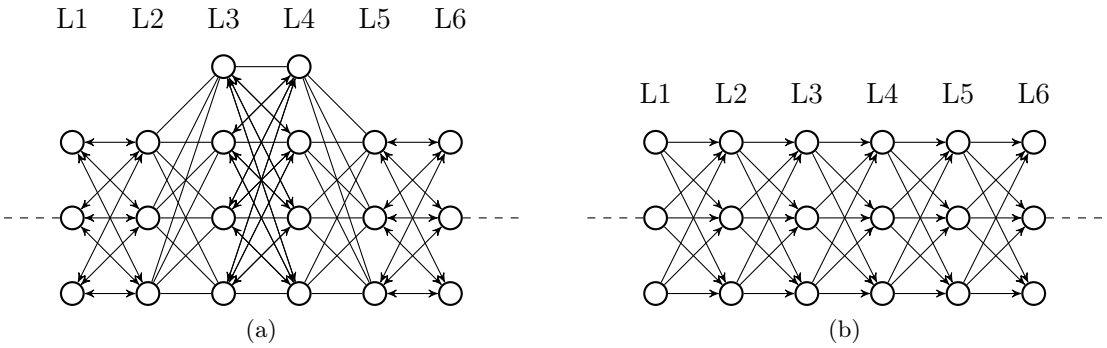
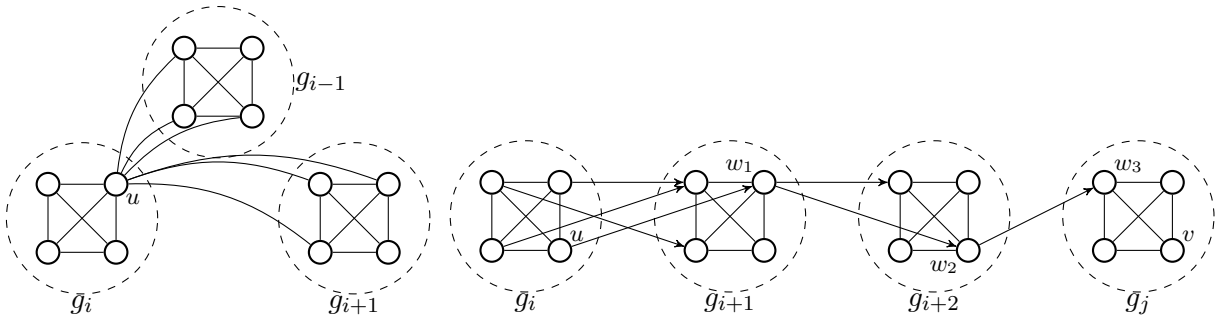


Figure 4.5: Layered networks.



(a) Connection of node. (b) The honest path between two honest nodes u and v in G_P.

Figure 4.6: Cluster-ring based network.

Cluster-ring-based networks [24, 25, 65, 82, 83]. The N nodes are clustered into $m = \sqrt{N}$ *ordered* groups, from g_0 to g_{m-1}. Each group is a clique. A node n in group g_i also links to a fixed-size set $\mathcal{S}_n$ of nodes in the next group ($\mathcal{S}_n \subset g_{i+1 \bmod m}$), and a fixed-size set $\mathcal{R}_n$ of nodes in the previous group ($\mathcal{R}_n \subset g_{i-1 \bmod m}$) where $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k + 1$. Thus, all groups virtually form a ring with g_0 being the successor of g_{m-1}.

In fact, this structure is a particular case of layered networks presented above in which each layer (group) is a clique of size $m = \sqrt{N}$, each node in one layer links to exact $2k + 1$ nodes in neighboring layer in which there is at least one honest friend, and all layers virtually form a ring (as depicted in Fig. 4.6).

In this case, the set of consumers of each node v_{in} in a group $g_i = \{v_{i,0}, v_{i,1}, \ldots, v_{i,m-1}\}$, (where $0 \leq i, n \leq m - 1$) might be determined as the output of the following function:

$$f : g_i \to 2^{g_{i+1 \bmod m}}$$
$$v_{in} \mapsto \{v_{i+1 \bmod m,(n+1) \bmod m}, v_{i+1 \bmod m,(n+2) \bmod m}, \ldots, v_{i+1 \bmod m,(n+2k+1) \bmod m}\}$$

We see that $|\mathcal{S}_{in}| = |f(v_{in})| = 2k + 1$ and the set of producers of a node $v_{in} \in g_i$ is $\mathcal{R}_{in} = \{v_{i-1 \bmod m,(n-1) \bmod m}, \ldots, v_{i-1 \bmod m,(n-2k-1) \bmod m}\}$ and $|\mathcal{R}_{in}| = 2k + 1$.

To be more clear, we will show the graph G_P in DPol [82, 83] is indeed a member of family $\mathcal{G}_2$. This graph satisfies property P_{g_1} with the sets of producers and consumers of each node are presented above. We will show this graph also implicitly fulfills the property P_{g_2} of family $\mathcal{G}_2$, i.e., G_P is an honest graph. Indeed, since an honest node links to at least one honest node in the neighboring group and each group is a clique, there exists an honest path connecting two honest nodes inside two consecutive groups. For two arbitrary honest nodes $u \in \mathcal{H}(g_i)$, $v \in \mathcal{H}(g_j)$

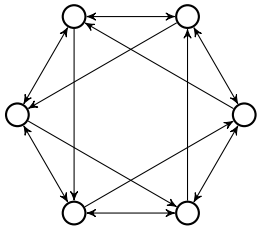


Figure 4.7: A circle-based network.

where $i < j$, by applying this result for a chain of two consecutive groups g_i and g_{i+1}, g_{i+1} and g_{i+2},..., g_{j-1} and g_j, we can see that there exists at least one honest path connecting u and v. For instance, Fig. 4.6b depicts a network of size $N = 16$. Since there are honest paths between u and w_1, w_1 and w_2, w_3 and v, there exists an honest path between u and v with intermediate honest nodes w_1, w_2, w_3. In conclusion, G_P is an honest graph. Noted that, for these kind of graphs, our protocol tolerates more dishonest nodes than DPol does [82, 83].

Circle-based networks. Consider a graph G_0 of size $N > 2k+1$ satisfying properties P_{g_2}, P_{g_3} of $\mathcal{G}_2$, and each node n has a set of consumers $\mathcal{S}_n$ which is the output of the function:

$$f\colon A \to 2^A$$
$$n \mapsto \{(n-1) \bmod N, (n+1) \bmod N, \ldots, (n+2k) \bmod N\}$$

where $A = \{0, 1, 2 \ldots, (N-1)\}$. Clearly, $|\mathcal{S}_n| = |f(n)| = 2k+1$. The set of producers is $\mathcal{R}_n = \{(n+1) \bmod N, (n-1) \bmod N, (n-2) \bmod N, \ldots, (n-2k) \bmod N\}$ of size $2k+1$. This follows that $d_n \geq 2k+1$ and so, G_0 also has property P_{g_1}. It infers $G_0 \in \mathcal{G}_2$. Fig. 4.7 depicts an example of this graph with $N = 6$ and $k = 1$ where an arrow from u to v depicts that node v is a consumer of u. In this graph, $\mathcal{S}_n \cap \mathcal{R}_n \neq \varnothing$.

Note that all protocols in [24,25,65,82,83] cannot be deployed in this graph since the condition $\mathcal{S}_n \cap \mathcal{R}_n = \varnothing$ (proposed in these works) is not satisfied in G_0. From this, we can conclude our protocol can be deployed on graph structures which are more general than the overlay structure used in [24, 25, 65, 82, 83].

4.4 Experimental evaluation

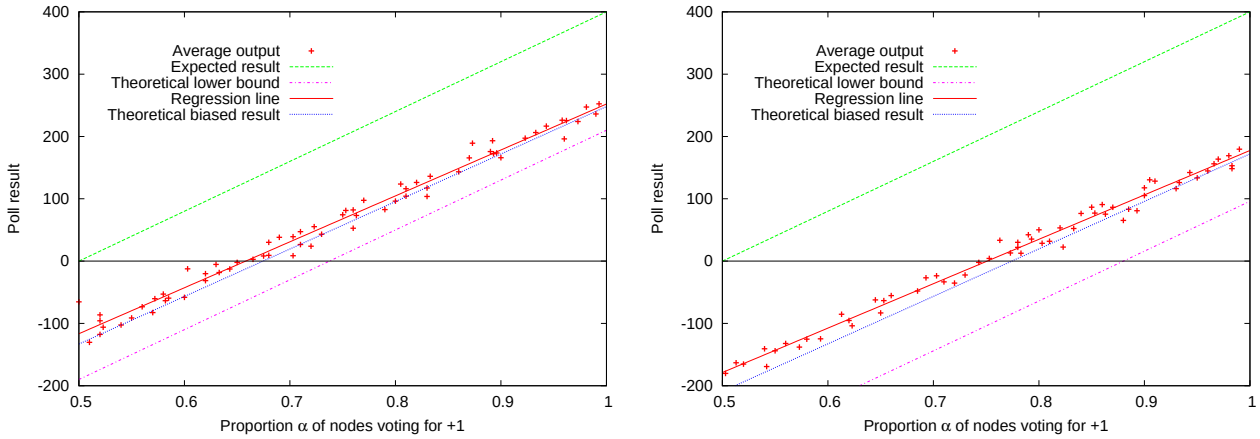
We do some experiments to analyze the correctness of the protocol by observing the difference between experiment output and the theoretical one.

Experimental setup

We implement protocol by using an open-source Java library, YALPS[9], to demonstrate the communication amongst node and facilitate the development and testing of the applications. YALPS supports for both asynchronous and synchronous models. The application based on YALPS framework can be run both in simulation and real-world mode. In the experiments, we use the synchronous model in message exchanging without crash or message loss. We conduct the experiments on the local machine with the following configuration: Intel Core 2 Quad CPU Q9550 @ 2.83GHz × 4 and Ubuntu OS 11.10.

We study the experiment with the existence of dishonest nodes, and examine whether the real impact from the dishonest coalition to the outcome is inside the analytical bound or not.

[9] http://yalps.gforge.inria.fr/



(a) $k = 1$ (b) $k = 2$

Figure 4.8: Experiment with $N = 400$, $D = 19$ to check the accuracy of protocol.

Moreover, we consider the worst case for the system in the sense that the dishonest nodes try to misbehave by expressing the attack to decrease the final result at most without being detected: each dishonest node always sends $2k+1$ shares of value "-1" in the sharing phase, and converts all receiving shares of "$+1$" into ones of "-1". Thus, each dishonest node affects the final result at most $2k+2$ in the sharing phase (if its desired vote is "$+1$" and it sends $2k+1$ shares of "-1") and $2(2k+1) = 4k+2$ in the broadcasting phase (if $2k+1$ receiving shares are "$+1$" and all are converted into "-1"). In other words, total impact will be up to $6k+4$. If we denote by α number of nodes voting "$+1$", then the expected result will be $\alpha N-(1-\alpha)N = (2\alpha-1)N$. So theoretically, the biased final outcome should be inside the interval $[(2\alpha-1)N-(6k+4)D; (2\alpha-1)N]$.

Results

We examine the same experiment condition about number of nodes and dishonest nodes as in [82, 83]. Without loss of generality, we consider α value in the interval $[0.5, 1.0]$.

Fig. 4.8 depicts our experiments for the network with $N = 400$, $D = 19$ in two subcases corresponding to two different values of privacy parameter $k = 1$ (in Fig. 4.8a) and $k = 2$ (in Fig. 4.8b). In each test, we compute the average output amongst all nodes and represented it as a point in the figures. We see that the experimental result is certainly inside two bounds: expected theoretical bound (thick-dashed line) and lower bound (dot-dashed line). The experiment result never touches the expected line as the dishonest node always sends $2k+1$ shares of "-1" and converts all receiving shares of "$+1$" to "-1" and this decreases the final output and makes it less than the expected outcome.

Moreover, the experiment result does not reach the theoretical lower bound either, i.e., the average impact is less than $6k+4$. The reason comes from the fact that the amount of average impact from dishonest nodes depends on the number of shares "$+1$" it receives from neighbors. Namely, in the first phase, nodes voting "$+1$" will send $k+1$ shares of "$+1$" and k shares of "-1", and vice versa. Therefore, the probability one node receives shares of "$+1$" from its neighbors is $\alpha.\frac{k+1}{2k+1}+(1-\alpha).\frac{k}{2k+1} = \frac{k+\alpha}{2k+1}$, and the average number of shares of "$+1$" is $(2k+1).\frac{k+\alpha}{2k+1} = k+\alpha$. From the position of the dishonest nodes, they convert all shares of "$+1$" into "-1". Thus, the impact will be $2(k+\alpha)$. Combining to the fact that they always distribute $2k+1$ shares of "-1" with the maximum impact is $2k+2$ (if its desired vote is "$+1$"), we achieve the total impact $2k+2+2(k+\alpha) = 4k+2\alpha+2$. In conclusion, with D dishonest nodes, the average biased

outcome is $(2\alpha-1)N-(4k+2\alpha+2)D$. In Fig. 4.8, we present this average value in a thin-dotted line.

We try to fit our data points with a regression line $a(2\alpha-1)-b(4k+2\alpha+2)$ and get these results (depicted as a solid line in Fig. 4.8): for $k=1$: $a=385$ and $b=17$, and for $k=2$: $a=373$ and $b=16$. These parameters are quite accurate compared to conditions of network with $N=400$ and $D=19$.

In Fig. 4.8, we see that the impact from the dishonest nodes in case $k=2$ is greater than in case $k=1$. This result is reasonable since we know that the higher value k is, the higher privacy can be hold but the higher impact dishonest nodes can enforce, and hence, the worse the final outcome is.

Besides, all nodes output the correct result (or the final result greater than 0) for $k=1$ when $\alpha \geq 0.67$, and for $k=2$ when $\alpha \geq 0.75$. It means the dishonest nodes confuse the majority of nodes for $k=1$ when $\alpha < 0.67$, and for $k=2$ when $\alpha < 0.75$. Compared to other recent polling protocols like [82, 83], that value of α in our experiment is similar to them.

4.5 Summary and discussion

In this chapter, we have proposed a design of a distributed polling protocol using synchronous model and defined a family of social graphs. We proved the structures of our family of graphs constitute necessary and sufficient condition to ensure privacy and accuracy properties of the protocol with the presence of dishonest nodes. To detect dishonest nodes' misbehaviors, we presented verification procedures by using routing table and shortest path scheme. Furthermore, a simple but useful technique based on shortest path scheme was introduced to prevent a node from receiving/sending so many duplicated messages without losing any necessary information. Unlike other works, we considered a protocol with a more general family of graphs, but obtained some similar results. More specifically, we achieved the same maximum number of votes that dishonest coalition can reveal, and the same impact from the coalition to the final output. Next chapter, we will describe an polling protocol which is based on asynchronous model and each node does not know the shortest path lengths.

Chapter 5

Asynchronous Model-based Polling Protocol

Contents

5.1 Polling model 67
5.1.1 Social interactions 67
5.1.2 Description of graph model 68
5.2 Polling protocol 69
5.3 Correctness 72
5.3.1 Protocol and graph without dishonest nodes 73
5.3.2 Protocol and graph with dishonest nodes 76
5.4 Experimental evaluation 80
5.5 Summary and discussion 82

In the previous contribution, we have developed a distributed polling protocol and a family of more general social graphs which ensures the correctness of the protocol and vote privacy of nodes. Nonetheless, the communication model is *synchronous* where all connection delays are bounded, and the system is driven with the presence of global clock.

In this chapter, we present a polling protocol operating in the *asynchronous* communication model (see Section 5.2) where the arrival order of messages is unpredictable. Moreover, this asynchronous version does not require any extra knowledge of the network (e.g., distance between two nodes) to perform the polling process. Given these two major differences, in this chapter, the broadcasting phase of the protocol is completely revised, as well as the correctness of the protocol are thoroughly analyzed (Section 5.3). Likewise the previous protocol, we also present the public verification scheme to detect dishonest nodes. Finally, we implement this new protocol and validate it with a performance evaluation (in Section 5.4).

5.1 Polling model

This section defines the user behaviors and presents the graph models to describe social networks.

5.1.1 Social interactions

The polling problem consists of a system with N uniquely identified nodes representing users of a social network. Each participant n expresses its opinion by giving a vote $v_n \in \{-1, 1\}$. After

collecting the votes of all nodes, the expected outcome is $\sum_n v_n$.

In this chapter, we consider the same assumptions given in Chapter 4 except the communication model is *asynchronous*. More precisely:

1. We consider the *asynchronous* model and the network contains no crash and message loss.
2. The message transmission in one edge takes one time unit.
3. The network includes honest and dishonest nodes.
4. Each node has at least one honest friend but it does not know exactly which friend is honest or not.
5. The dishonest nodes want to misbehave to achieve these goals without affecting their reputation and to be tagged in their profiles: (i) bias the result of the election by promoting their votes or changing the values they received from other honest nodes; (ii) infer the opinions of other nodes.
6. In order to unify the opinions and not give compensating effects, all dishonest nodes make the single coalition $\mathcal{D}$ of size D and give the same corrupted values.
7. The dishonest nodes are rather restricted than Byzantine nodes.
8. We do not take into account the Sybil attacks, spam and the situation that dishonest nodes wrongly blame honest ones.
9. All nodes have to send/receive/forward messages without delaying if they are requested.

5.1.2 Description of graph model

We present the social network in the form of models of social graphs as introduced in Section 3.2.1 of Chapter 3.

Moreover, like the model in the previous chapter, each node n maintains a set of direct neighbors $\Gamma(n)$ (or Γ_n) of size d_n. In addition, it also holds other two subsets of $\Gamma(n)$: a set $\mathcal{S}_n$ of *consumers* containing nodes that n sends messages to, and $\mathcal{R}_n$ of *producers* relating to nodes for which n acts as a consumer. They might not be disjoint, i.e., $\mathcal{S}_n \cap \mathcal{R}_n \neq \varnothing$, as depicted in Fig. 4.1.

Like [82, 83], we use a predefined parameter $k \in \mathbb{N}$ and $k \leq \lfloor(-1 + \sqrt{3N+1})/3\rfloor$ (this parameter will be detailed in section 5.2) to present the features of our social graphs. Let $G = (V, E)$ be a social graph with the following properties:

Property 5.1 (P_{g_1})**.** *$d_n \geq 2k+1$ and $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k+1$, for every $n \in V$.*

Property 5.2 (P_{g_2})**.** *G is an honest graph, i.e., for every honest nodes u, v, there exists a path $p(u, v)$ containing only intermediate honest nodes.*

Property 5.3 (P_{g_3})**.** *$D \leq N/10$.*

According to Property P_{g_1}, a set of consumers and a set of producers of one node have the same size and might not be disjoint. Property P_{g_2} ensures each honest node always obtains one correct

version of data from other honest ones. Property P_{g_3} enables us to limit the control of dishonest users in the whole system.

From these properties, we characterize two families of graphs (which are similar to the ones presented in Chapter 4):

(i) $\mathcal{G}_1 = \{G \mid \mathcal{D}(G) = \varnothing$ and G satisfies $P_{g_1}\}$.

(ii) $\mathcal{G}_2 = \{G \mid \mathcal{D}(G) \neq \varnothing$ and G satisfies P_{g_1}, P_{g_2} and $P_{g_3}\}$.

Graphs in $\mathcal{G}_1$ contain only honest nodes and satisfy property P_{g_1}. Graphs in $\mathcal{G}_2$ contain honest and dishonest nodes and satisfy properties P_{g_1}, P_{g_2} and P_{g_3}.

5.2 Polling protocol

Generally, the polling protocol includes three phases (see Algorithm 2): (*i*) *Sharing*, (*ii*) *Broadcasting* and (*iii*) *Aggregating*. Phase *Sharing* describes the generation, distribution of a set of shares of each node to its neighbors as well as collecting these shares from its neighbors. In the *Broadcasting* phase, each node broadcasts messages containing the total shares, which are collected in the Sharing phase, to its direct and indirect neighbors. The last phase, *Aggregating*, shows the process that each node decides data received from other nodes and computes the final outcome.

Sharing. In this phase, each node n contributes its opinion by sending a set of shares expressing its vote $v_n \in \{-1, 1\}$ to its consumers. Namely, n generates $2k + 1$ shares $\mathcal{M}_n = \{m_1, m_2, ..., m_{2k+1}\}$ where $m_i \in \{-1, 1\}$, $i = 1, 2, ..., 2k + 1$ including: $k + 1$ shares of value v_n, and k shares of opposite v_n's value. The intuition of this creation is to regenerate the vote v_n when the shares are summed. Later it randomly generates a permutation μ_n of $\mathcal{M}_n$, and sends shares to $2k + 1$ consumers. Lines 4–9 in Algorithm 2 describe this activity. Node also receives exactly $2k + 1$ shares from its producers. Note that $\mathcal{S}_n$ and $\mathcal{R}_n$ might not be disjoint.

After each node collects $2k + 1$ shares from its neighbors, and sums into *collected data* c_n (lines 10–15 in Algorithm 2), this phase is complete. Fig. 5.1 illustrates an example of the protocol for $k = 1$. Fig. 5.1a presents desired vote of each node, whereas Fig. 5.1b depicts the sharing phase at node A. Node A would like to vote $+1$. Thus, it generates a set of $2k + 1 = 3$ shares $\{+1, -1, +1\}$ which total equals to $v_A = 1$. Fig. 5.1c shows node A collects the shares from its producers and computes the collected data $c_A = 3$.

Broadcasting. This phase operates in *cycles*. Briefly, at the ith cycle ($i \geq 1$), a node n carries out four steps as follows: This phase operates in *cycles*. Briefly, at the ith cycle ($i \geq 1$), each node n performs the following four steps:

(B1) It *starts* a cycle i by sending its direct neighbors *data messages* that contains the collected data of neighbors at distance $(i - 1)$ from it.

(B2) It *waits* for the receipt of data messages of cycle i from all direct neighbors.

(B3) It sends *acknowledgement messages* of cycle i to all direct neighbors.

(B4) It *waits* for the receipt of acknowledgement messages of cycle i from all direct neighbors. It *finishes* this cycle once it has examined the received information and prepared the data for the next cycle.

Algorithm 2: ASYNCHRONOUS POLLING ALGORITHM AT NODE $n \in \{0, 1, ..., N-1\}$

Input:
v_n: A vote of node n, value in $\{-1, 1\}$
k : privacy parameter

Variables:
c_n: collected data, $c_n = 0$
$\mathcal{C}_n$: set of possible collected data
$\mathcal{C}_n[\{0, 1, ..., N-1\} \to \varnothing]$
h_n: set of final deciding collected data
$h_n[\{0, 1, ..., N-1\} \to \perp]$
$\mathcal{T}_n$: routing table
$\mathcal{T}_n[\{0, 1, ..., N-1\} \to \varnothing]$
$\mathcal{Z}_n$: sending data
$\mathcal{Z}_n[\{0, 1, ..., N-1\} \to \varnothing]$

Output: result

Algorithm

1 `Sharing`(v_n)
2 `Broadcasting`(c_n)
3 `Aggregating`()

Procedure `Sharing`(v_n)

4 $\mathcal{M}_n \leftarrow \{v_n\}$
5 **for** $i \leftarrow 1$ **to** k **do**
6 $\quad \mathcal{M}_n \leftarrow \mathcal{M}_n \cup \{v_n\} \cup \{-v_n\}$
7 $\mu_n \leftarrow_{rand} \mathcal{M}_n$
8 **for** $i \leftarrow 0$ **to** $2k$ **do**
9 $\quad$ send (SHARE, $\mu_n[i]$) to $\mathcal{S}_n[i]$
10 $count \leftarrow 0$
11 **while** ($count < |\mathcal{R}_n|$) **do**
12 $\quad$ **upon event** (receiving (SHARE, m) from neighbor r in the first time) **do**
13 $\quad\quad$ **if** ($r \in \mathcal{R}_n$ **and** $m \in \{-1, 1\}$) **then**
14 $\quad\quad\quad c_n \leftarrow c_n + m$
15 $\quad\quad\quad count \leftarrow count + 1$

Procedure `Broadcasting`(c_n)

16 $count \leftarrow 0$
17 $i \leftarrow 1$
18 **while** (true) **do**
19 $\quad$ **foreach** ($r \in \Gamma(n)$) **do**
20 $\quad\quad$ **if** ($i = 1$) **then** send (DATA, n, i, $\{(n, c_n)\}$) to r
21 $\quad\quad$ **else** send (DATA, n, i, $\mathcal{Z}_n[r]$) to r
22 $\quad\quad \mathcal{Z}_n[r] \leftarrow \varnothing$
23 $\quad$ Wait until all messages of type (DATA, $*$, i, $\mathcal{Z}_*[n]$) are received from all direct neighbors and,
24 $\quad$ **upon event** (receiving message (DATA, t, i, $\mathcal{Z}_t[n]$) from direct neighbor t) **do**
25 $\quad\quad$ **if** `RecDataEvt`(t,i,$\mathcal{Z}_t[n]$, $count$) = false **then** break
26 $\quad$ **foreach** ($r \in \Gamma(n)$) **do**
27 $\quad\quad$ send (ACK, n, i) to r
28 $\quad$ **if** ($|\Gamma(n)| = N-1$) **then** break
29 $\quad$ Wait until all messages of type (ACK, $*$, i) are received from all direct neighbors
30 $\quad i \leftarrow i + 1$

Function `RecDataEvt`(t, i, $\mathcal{Z}_t[n]$, $count$)

31 **if** ($\mathcal{Z}_t[n] = \varnothing$) **then**
32 $\quad count \leftarrow count + 1$
33 $\quad$ **if** ($count = |\Gamma(n)|$) **then return** false
34 $\quad$ **else return** true
35 **foreach** (pair (s, c_s) in $\mathcal{Z}_t[n]$) **do**
36 $\quad$ **if** ($s = n$) **then** continue
37 $\quad$ **if** ($c_s \notin \mathcal{C}_n[s]$) **then**
38 $\quad\quad \nu_s \leftarrow c_s$
39 $\quad\quad \mathcal{C}_n[s] \leftarrow \mathcal{C}_n[s] \cup \{c_s\}$
40 $\quad\quad$ Store (s, c_s) into $\mathcal{Z}_n[w]$ for each neighbor $w \neq t$
$\quad$ **else**
41 $\quad\quad \nu_s \leftarrow \perp$
42 $\quad \mathcal{T}_n[s] \leftarrow \mathcal{T}_n[s] \cup \{(t, c_s, \nu_s, i)\}$
43 **return** true

Procedure `Aggregating`()

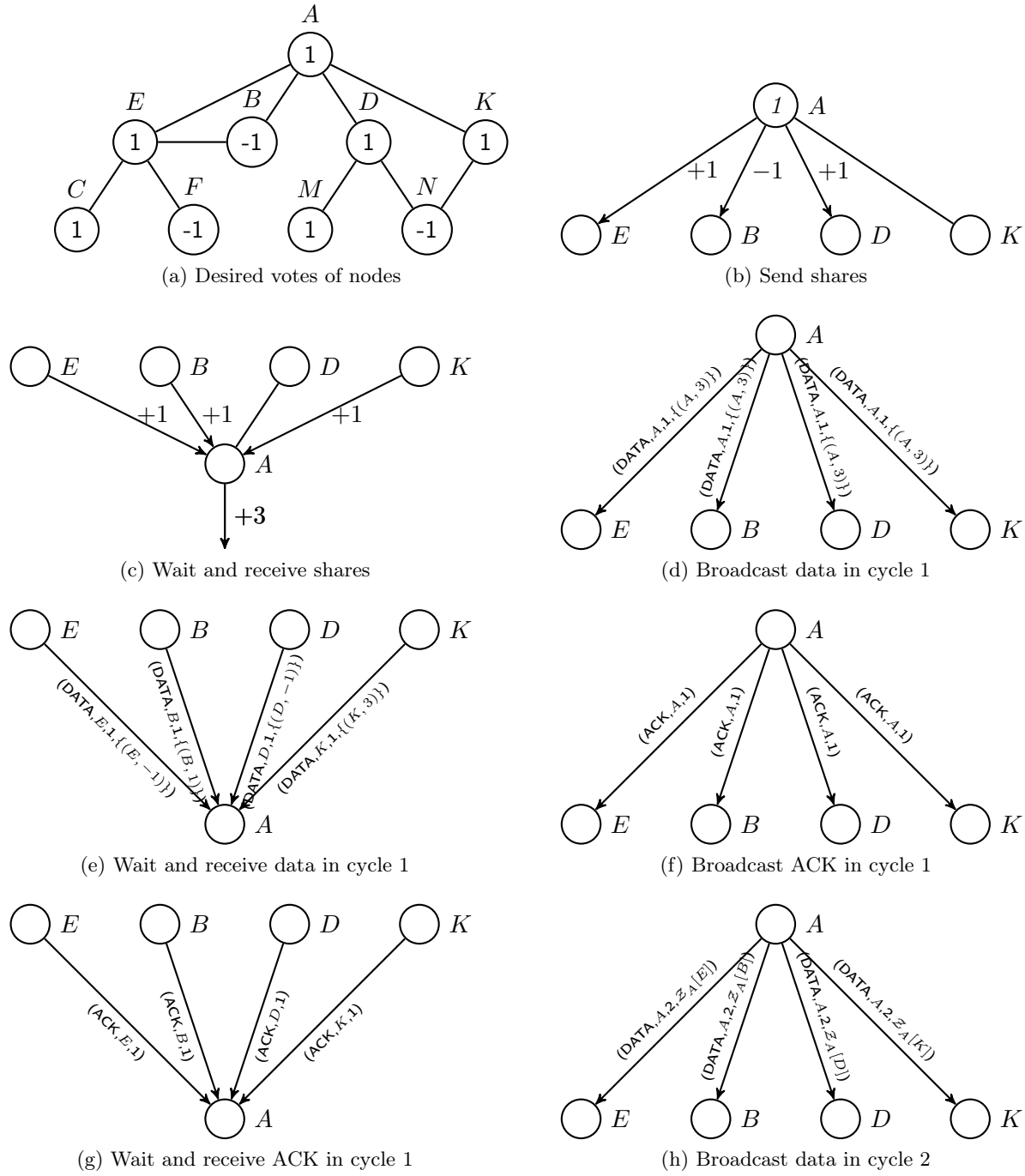
44 result $\leftarrow c_n$
45 **for** $s \leftarrow 0$ **to** $N-1$ **do**
46 $\quad$ **if** ($s \neq n$) **then**
47 $\quad\quad h_n[s] \leftarrow$ `CheckInconsistency`(s)
48 $\quad\quad$ result $\leftarrow$ result $+ h_n[s]$

Procedure `CheckInconsistency`(s)

49 **if** ($|\mathcal{C}_n[s]| = 1$) **then return** $\mathcal{C}_n[s][0]$
50 **return** *correct value after verifying* $\mathcal{T}[s]$ *of neighbors*

More specifically, at cycle i, firstly, node n sends all direct friends data messages of the type (DATA, $n, i, \mathcal{Z}_n$)[10] indicating identity of node n, cycle i, and a set $\mathcal{Z}_n$ of pairs including collected data and identities of nodes at distance $(i-1)$ from it (lines 19–22). Upon receiving a data

[10]If this message is corrupted, we can detect this misbehavior. See Lemma 5.7 for the detail of this attack.



(a) Desired votes of nodes

(b) Send shares

(c) Wait and receive shares

(d) Broadcast data in cycle 1

(e) Wait and receive data in cycle 1

(f) Broadcast ACK in cycle 1

(g) Wait and receive ACK in cycle 1

(h) Broadcast data in cycle 2

Figure 5.1: Polling algorithm for $k = 1$.

message (DATA, $t, i, \mathcal{Z}_t$) from neighbor t, node n performs the following actions (see procedure `RecDataEvt`() in Algorithm 2):

1. *Termination detection:* If the set $\mathcal{Z}_t[n]$ is nonempty, it does next action. Otherwise, if it has already obtained all empty data sets from all direct neighbors, node n stops this phase (lines 25 and 31–34).

2. *Data analyzing:* Node examines information in the data messages from its neighbors by checking each pair (s, c_s) in the set $\mathcal{Z}_t[n]$ (lines 35–43). If the pair contains its own data, it simply ignores that pair. Otherwise n checks $\mathcal{C}_n[s]$, a set of possible values of node with identity s, to determine whether c_s is already presented in it: If $\mathcal{C}_n[s]$ does not contain c_s, node n will add it into $\mathcal{C}_n[s]$. It also prepares its data messages to send to neighbor $w \neq t$ for the $(i+1)$th cycle by identifying all collected data and identities of nodes at distance i it has just learned and storing them in the set $\mathcal{Z}_n[w]$ (line 40). Moreover, as the information can be changed by intermediate dishonest nodes, each node n also keeps up the information from source s in the *routing table* $\mathcal{T}_n[s]$ which is used in the verification process later. This table contains the following fields: neighbor identity of the receiving message (e.g., t), receiving data (e.g., c_s), forwarding data (e.g., ν_s), cycle (i.e., i).

When n gets data messages from all its direct neighbors, it broadcasts message (ACK, n, i) in order to inform them that it already obtains all data messages at the ith cycle (lines 26–27). It then waits for acknowledgement messages of cycle i from all of its neighbors before doing next cycle $i+1$.

This phase starts with each node sending its own collected data to all its neighbors saying that it is at distance 0 from itself. A node keeps on transmitting messages at cycle i as long as it received (and not dropped) a data message containing the set $\mathcal{Z}_t \neq \varnothing$ from a certain neighbor t at cycle $i-1$. More precisely, a node stops transmitting: (i) at cycle $i > 1$ if until cycle i, it has obtained empty sets $\mathcal{Z}_t$ from all of its direct neighbors (lines 31–34); or (ii) at the first cycle if it has $N-1$ direct neighbors (line 28).[11]

For instance, Figures 5.1d–5.1h depict this phase for node A in the first two cycles. In cycle 1, node A sends messages containing its own collected data to all its neighbors (Fig. 5.1d), then it waits for their data (Fig. 5.1e). After receiving all data messages in cycle 1, node A sends acknowledgement messages to all its friends (Fig. 5.1f) and then waits for receiving their acknowledgements (Fig. 5.1g). Cycle 2 starts when it gets all acknowledgement messages from its friends (Fig. 5.1h).

Aggregating. In this phase, a node n has to decide on the collected data of other nodes before computing the final result. To make decision for the one of node s, it checks $|\mathcal{C}_n[s]|$ (lines 49–50 in Algorithm 2): if $|\mathcal{C}_n[s]| = 1$, the single element in $\mathcal{C}_n[s]$ is chosen as a correct collected data; otherwise there exists an inconsistency and it should do the verification (this verification will be explained in Lemma 5.7). By doing this, n gets the correct collected data of source s. So, in any case, n achieves the correct copy of collected data of source s. It then stores that value as one item $h_n[s]$ in the array h_n, which contains the chosen collected data of other nodes, and adds into result (lines 44–48 in Algorithm 2). After checking and summing up all collected data of nodes (including its own collected data c_n), n obtains the final result (that is $\text{result} = c_n + \sum_{i \neq n} h_n[i]$).

5.3 Correctness

In this section, we first we analyze its correctness and complexities when deployed for graphs of $\mathcal{G}_1$ and $\mathcal{G}_2$. Then we show those properties are necessary and sufficient conditions to ensure the correctness of our protocol in graphs without and with the presence of dishonest nodes. The

[11] We later show that, at the worst case scenario, a node stops transmitting messages only after $Diam(G)$ (resp. $\max\{Diam(G), N-1-D\}$) cycles end when the system contains no (resp. D) dishonest nodes in Theorem 5.5 (resp. Theorem 5.7).

properties of the polling protocol are introduced in Sub-section 4.3.1 of Chapter 4 and we omit that part here.

5.3.1 Protocol and graph without dishonest nodes

In this section, we consider only graphs G of family $\mathcal{G}_1$ and analyze the correctness (including accuracy and termination) of our protocol when deployed for these graphs. Next we give spatial, message and time complexities. Finally, we show the properties of $\mathcal{G}_1$ are necessary and sufficient conditions to ensure the correctness of our protocol. It is noted that we do not consider the privacy property as there is no dishonest nodes in this case.

Accuracy

Theorem 5.1 (Accuracy)**.** *Consider a polling system of size N with only honest nodes where each node n expresses a vote v_n. The polling algorithm is guaranteed that each node outputs accurate expected output $\sum_{n=0}^{N-1} v_n$.*

Proof. In the sharing phase, each node n sends a set of shares $\mathcal{M}_n = \{m_{n_1}, m_{n_2}, ..., m_{n_{2k+1}}\}$ to its consumers where $\sum m_{n_i} = (k+1).v_n + k.(-v_n) = v_n$, and also receives a set of shares $\{m'_{n_1}, m'_{n_2}, ..., m'_{n_{2k+1}}\}$ from its $2k+1$ producers to obtain a collected data $c_n = \sum_{j=1}^{2k+1} m'_{n_j}$. Without the presence of dishonest nodes, crashes and message losses, each message from the source successfully reaches the destination, and thus the set of all sending shares of all nodes will be exactly coincided with the set of all receiving shares of all nodes, namely:

$$\bigcup_{V}\{m_{n_1}, m_{n_2}, ..., m_{n_{2k+1}}\} = \bigcup_{V}\{m'_{n_1}, m'_{n_2}, ..., m'_{n_{2k+1}}\}$$

In the broadcasting phase, each node n broadcasts its collected data to its direct neighbors in the first cycle, then they do honestly forward that value to neighbors of neighbors of n in the second cycle and so on. Each node's data is finally arrived to all other ones. Indeed, a node keeps on transmitting messages at cycle $i+1$ as long as it received (and not dropped) a data message containing a set $\mathcal{Z}_t \neq \varnothing$ from a direct neighbor t at cycle i. The set $\mathcal{Z}_t \neq \varnothing$ indicates that its neighbor t has friends at distance $i-1$ from t and it had received some new information of some other node s in cycle $i-1$, thus node n should continue sending messages to other friends at cycle $i+1$ (if that information does not exist in $\mathcal{C}_n[s]$). In contrast, $\mathcal{Z}_t = \varnothing$ means there is no friend of t at distance $\geq i-1$ from t, hence, n should stop waiting for messages from t. Thus, if a node stops this phase after cycle i ends, it is ensured to receive all collected data from all nodes in the network.

Thus, an array h_n contains all collected data of all nodes in the system and these values come from all the receiving shares of all the nodes. Consequently, the final computation gives us the value:

$$\mathsf{result} = c_n + \sum_{\substack{0\leq i<N \\ i\neq n}} h_n[i] = \sum_{i=0}^{N-1} c_i = \sum_{i=0}^{N-1}\sum_{j=1}^{2k+1} m'_{i_j} = \sum_{i=0}^{N-1}\sum_{j=1}^{2k+1} m_{i_j} = \sum_{i=0}^{N-1} v_i$$

□

Termination. In this part, we will show our protocol is ensured to finally terminate. We say: (i) a node n *starts* (resp. *finishes*) cycle i if and only if n already sends (resp. receives) data (resp. acknowledgement) messages (DATA,$n, i, \mathcal{Z}_n$) (resp. (ACK, *, i)) to (resp. from) all its direct friends; (ii) a node n is *blocked* in cycle i if and only if n is doing step (B2) or (B4), i.e.,

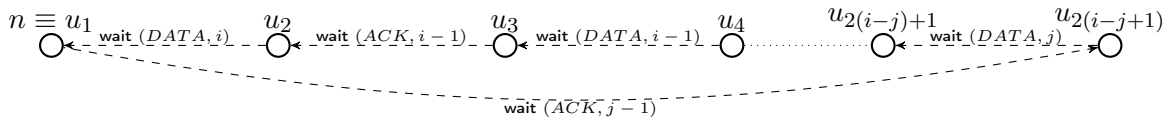


Figure 5.2: Deadlock amongst nodes $n \equiv u_1$, u_2,..., $u_{2(i-j+1)}$.

waiting for data messages or acknowledgement messages from some direct friend; and (iii), a node n is *in* cycle i if and only if n is doing some step of cycle i.

Lemma 5.1. *If a node n is in step (B2) of cycle i then each direct neighbor of n is either in cycle i or $i-1$.*

Proof. Since node n already finished cycle $i-1$, it must have got $(ACK, u, i-1)$ from all direct neighbors u. This implies they all terminate step (B3) of cycle $i-1$, i.e., they are at least in step (B4) of cycle $i-1$. Moreover, n has not sent a message (ACK, n, i), thus, all its direct friends could not finish cycle i. From this, we have the claim of this lemma. □

Lemma 5.2. *If a node n is in step (B4) of cycle i then each direct neighbor of n is either in cycle i or $i+1$.*

Proof. As a node n already did step (B3) of cycle i, i.e., received messages $(DATA, u, i)$ from all friends $u \in \Gamma(n)$, it infers all its friends already started cycle $\geq i$. In addition, no friend of n receives $(DATA, n, i+1)$ because n has not finished cycle i. Thus, they could not accomplish step (B2) of cycle $i+1$, in other words, they are in cycle $\leq (i+1)$. This yields the proof of lemma. □

Theorem 5.2. *If a node n is blocked in cycle i then each direct neighbor of n is either in cycle $i-1$, or i, or $i+1$.*

Proof. The statement is hold by Lemmas 5.1 and 5.2. □

Corollary 5.1. *If a node n is blocked in cycle i then its neighbors at distance j (where $0 \leq j < i$) from n are in cycle varied in the range between $i-j$ and $i+j$.*

Proof. The proof is given by induction with the basic case like the proof in Theorem 5.2. □

Lemma 5.3. *If a node n is in cycle i then it eventually progresses to cycle $i+1$.*

Proof. Assume contrary, i.e., there is a deadlock: n is blocked forever at cycle i because it is waiting for receiving a data (or acknowledgement) message of some node in cycle i, and there is also other node waiting for the progress of node n to receive n's acknowledgement (or data) message of some cycle $j-1$ (or j) for $i > j > 1$. In other words, w.l.o.g., there is a loop as depicted in Fig. 5.2: a node $n \equiv u_1$ is waiting for a message $(DATA, u_2, i)$ from certain node $u_2 \in \Gamma(u_1)$, node u_2 is waiting for a message $(ACK, u_3, i-1)$ from other node $u_3 \in \Gamma(u_2)$ and $u_3 \in \Gamma^2_{u_1}$. Similarly, u_3 must have waited a message $(DATA, u_4, i-1)$ from node $u_4 \in \Gamma(u_3)$ and $u_4 \in \Gamma^3_{u_1}$, and so on. Node $u_{2(i-j)+1}$ is waiting for a message $(DATA, u_{2(i-j+1)}, j)$ from some node $u_{2(i-j+1)} \in \Gamma^{2(i-j)+1}_{u_1}$, and node $u_{2(i-j+1)}$ must have waited for message $(ACK, u_1, j-1)$. We see that, u_1 already finished cycle $i-1 > j-1$, thus, it already sent $(ACK, u_1, j-1)$ to all neighbors and $u_{2(i-j+1)}$ must have received it. Therefore the loop does not exist. Contradiction! □

Theorem 5.3. *If a node n is in step (B2) of cycle i then there exist some neighbors at distance $2j$ doing step (B2) of cycle $(i-j)$ and some neighbors at distance $2j+1$ doing step (B4) of cycle $(i-j-1)$ where $0 \leq j < i$.*

Proof. The proof is by induction on j. For $j = 0$, by Lemma 5.1 there exists a direct neighbor is in step (B4) of cycle $i-1$ and we are done. Let us suppose that the statement of the theorem holds for some $j \geq 0$ (and $j < i-1$). It means there exists a node $u \in \Gamma_n^{2j+1}$ such that u is waiting for receiving a message $(ACK, n_{2j+2}, i-j-1)$ from some node $n_{2j+2} \in \Gamma(u)$. Note that, by Lemma 5.3, n_{2j+2} could not be at distance less than $2j+1$ from n since otherwise there will be a deadlock similar to Fig. 5.2 and we already shown this deadlock does not exist. Thus, node $n_{2j+2} \in \Gamma_n^{2j+2}$. As u is waiting for receiving a message $(ACK, n_{2j+2}, i-j-1)$, n_{2j+2} must have waited for a message $(DATA, n_{2j+3}, i-j-1)$ from some node n_{2j+3} where $n_{2j+3} \in \Gamma(n_{2j+2})$, $n_{2j+3} \neq u$ and $n_{2j+3} \in \Gamma_n^{2j+3}$, i.e., n_{2j+2} is in step (B2) of cycle $i-j-1$. That also infers there exists a node $n_{2j+4} \in \Gamma(n_{2j+3})$ and $n_{2j+4} \neq n_{2j+2}$ such that n_{2j+3} is waiting for a message $(ACK, n_{2j+4}, i-j-2)$ from n_{2j+4}, in other words, n_{2j+3} is in step (B4) of cycle $i-j-2$. Therefore, the statement is correct for $j+1$. □

Theorem 5.4. *If a node n is in step (B4) of cycle i then there exist some neighbors at distance $2j+1$ doing step (B2) of cycle $(i-j)$ and some neighbors at distance $2j+2$ doing step (B4) of cycle $(i-j-1)$ where $0 \leq j < i$.*

Proof. The proof is similar to the one of Theorem 5.3. □

Theorem 5.5 (Termination)**.** *The polling protocol is ensured to eventually terminate.*

Proof. In the sharing phase, each node n has to send and receive a finite number $(2k+1)$ of messages.

In the broadcasting phase, a node keeps on transmitting messages at cycle $i+1$ as long as it received (and not dropped) a data message containing a set $\mathcal{Z}_t \neq \varnothing$ from a direct neighbor t at cycle i. The set $\mathcal{Z}_t \neq \varnothing$ indicates that its neighbor t has friends at distance $i-1$ from t and it had received some new information of some other node s in cycle $i-1$, thus node n should continue sending messages to other friends at cycle $i+1$ (if that information does not exist in the set $\mathcal{C}_n[s]$). In contrast, $\mathcal{Z}_t = \varnothing$ means there is no friend of t at distance $\geq i-1$ from t, hence, n should stop waiting for messages from t. At the worst case scenario, its farthest friends are at distance $Diam(G)$ and thus it stops this phase only after $Diam(G)$ cycles end. It is also noted that, if a node n has $N-1$ friends: n stops transmitting or receiving data after the first cycle.

So, the number of cycles of the broadcasting phase is limited. And in each cycle, each node n sends/receives exactly d_n data messages and d_n acknowledgement messages. Moreover, by Lemma 5.3 the protocol does not have deadlock, and there is no message loss or crash, each phases completes. The algorithm has a finite number of phases, by Definition 4.3 the protocol is guaranteed to finally terminate. □

Asymptotic complexity. We analyze the spatial, message and time complexities of the protocol in Propositions 5.1–5.3.

Proposition 5.1 (Spatial complexity)**.** *The total space each node n must hold is $\mathcal{O}(k + Nd_n)$.*

Proof. Each node n needs to maintain the set of $2k+1$ consumers, the set of $2k+1$ producers, the list of d_n direct neighbors to contact, the set $\mathcal{Z}_n$ to store pairs of identity and collected data

for friends, the set $\mathcal{C}_n$ of possible collected data of other nodes, the routing table $\mathcal{T}_n$, and the set h_n to store the final choosing collected data $(N-1)$.

For $\mathcal{Z}_n$: in cycle $i = 1$, it receives one value from a direct neighbor u, and this information is then stored in $\mathcal{Z}_n[w]$ for each direct neighbor $w \neq u$ which takes space of size $d_n - 1$. With d_n direct neighbors, this requires the space of size $(d_n - 1)d_n$. At the worst case scenario, a node stops transmitting messages (in the broadcasting phase) only after $Diam(G)$ cycles end. In each cycle $2 \leq i \leq Diam(G)$, n sends at most $N-2$ values for one neighbor u and also receives the same amount of values from u. Therefore, it needs $(N-2)d_n$ units of storage. This implies, the total space for storing $\mathcal{Z}_n$ in the protocol is $\max\{(d_n-1)d_n, (N-2)d_n)\} = \mathcal{O}(Nd_n)$.

Similarly, for $\mathcal{T}_n$: the worst case for the routing table $\mathcal{T}_n[s]$ is when all shortest paths between s and n are passed by its friends. In other words, $\mathcal{T}_n[s]$ contains at most d_n rows. It infers that $|\mathcal{T}_n| \leq (N-1).d_n$.

As there is no dishonest node, during the broadcasting phase, node n just inserts into the set $\mathcal{C}_n[s]$ one value for each node s, i.e., $|\mathcal{C}_n[s]| = 1$, and thus $|\mathcal{C}_n| = N-1$. Therefore, the spatial complexity in this case is $\mathcal{O}(k) + \mathcal{O}(d_n) + \mathcal{O}(Nd_n) + \mathcal{O}(N-1) = \mathcal{O}(k + Nd_n)$. □

Proposition 5.2 (Message complexity)**.** *Each node n sends $\mathcal{O}(k + Diam(G).d_n)$ messages.*

Proof. In the sharing phase, a node n sends $2k+1$ messages to its consumers. In the broadcasting phase, it sends $2d_n$ data and acknowledgement messages to all of its direct neighbors. There are at most $Diam(G)$ cycles. Accordingly, the message complexity is $\mathcal{O}(2k+1)+\mathcal{O}(2d_n.Diam(G)) = \mathcal{O}(k + Diam(G).d_n)$. □

Proposition 5.3 (Time complexity)**.** *Assume time evolves in rounds, i.e., each message transmission incurs a delay of at most one round. Then the protocol operates in $\mathcal{O}(k+Diam(G).\max_n\{d_n\})$rounds.*

Proof. The sharing phase operates in $2k+1$ rounds. In the broadcasting phase, a node n sends $2d_n$ messages to direct neighbors, and all the nodes broadcast their collected data in parallel, so this activity takes at most $\max_n\{d_n\}$ rounds. Node has to repeat at most $Diam(G)$ cycles. Therefore, the time complexity is $\mathcal{O}(2k+1)+\mathcal{O}(\max_n\{2d_n\}.Diam(G)) = \mathcal{O}(k+Diam(G).\max_n\{d_n\})$. □

The necessary and sufficient conditions of graph without dishonest nodes

In the following part, we examine the necessary and sufficient conditions for our protocol to be deployed successfully in the ideal case which contains no dishonest nodes.

Theorem 5.6. *The properties of $\mathcal{G}_1$ are the necessary and sufficient conditions for the polling protocol to be deployed correctly in the system without dishonest nodes.*

Proof. The proof is similar to Theorem 4.2 of Chapter 4 □

5.3.2 Protocol and graph with dishonest nodes

In this section, we revisit the relation between protocol and graph, but approach it with the presence of D dishonest nodes. We consider graphs G of family $\mathcal{G}_2$ and analyze the correctness (including privacy, accuracy and termination) of our protocol when deployed for these graphs. Finally, we show the properties of $\mathcal{G}_2$ are necessary and sufficient conditions to ensure the correctness of our protocol.

Termination. We prove that our protocol is guaranteed to finally terminate in the following Theorem.

Theorem 5.7 (Termination)**.** *The polling protocol is ensured to eventually terminate with the presence of D dishonest nodes.*

Proof. In the sharing phase, each node n has to send and receive a finite number $(2k+1)$ of messages.

In the broadcasting phase, a node keeps on transmitting messages at cycle $i+1$ as long as it received (and not dropped) a data message containing the set $\mathcal{Z}_t \neq \varnothing$ from a direct neighbor t at cycle i. The set $\mathcal{Z}_t \neq \varnothing$ indicates that its neighbor t received some new information of some other node s at cycle $i-1$, and node n transmits that information if it does not exist in $\mathcal{C}_n[s]$.

In the worst case, at each cycle, a node n always obtains a set $\mathcal{Z}_t \neq \varnothing$ from some direct neighbor t which contains a value of some source s not existing in the set $\mathcal{C}_n[s]$. As we know, each dishonest node can corrupt the received data from a certain source node by sending different values. Thus, a set $\mathcal{C}_n[s]$ of node n could not contain more than $D+1$ different values of source s. This infers, a node n stops transmitting data of some source s after receiving the data passed by one path amongst two following paths which has higher length: the longest honest path or the shortest path (may contain dishonest nodes) between n and s. Note that, since the network G is honest and contains $D < N/2$ dishonest nodes, the maximum length of an honest path is $N-1-D$ (and $N-1-D \geq D$), and the maximum length of all shortest paths is $Diam(G)$. Therefore, at the worst case scenario, a node is guaranteed to stop transmitting until $\max\{Diam(G), N-1-D\}$ cycles end. That infers the number of cycles of the broadcasting phase is limited.

In each cycle, each node n sends/receives exactly d_n data messages and d_n acknowledgement messages. Moreover, by Lemma 5.3 the protocol does not have deadlock, and there is no message loss or crash, each phases completes. The algorithm has a finite number of phases, by Definition 4.3 the protocol is guaranteed to finally terminate. □

Privacy. We omit this part as it is analyzed similarly to the one in Section 4.3.3 of Chapter 4.

Accuracy. We present the accuracy based on the ability of honest nodes to get correct output and to control the impact from dishonest nodes. First we discuss the capabilities of dishonest nodes to affect the accuracy in the following definition.

Definition 5.1 (Dishonest capabilities)**.** *A dishonest node may affect the poll outcome with the following* misbehaviors*:*

1. *In the sharing phase, it sends more than $k+1$ (but not greater than $2k+1$) identical shares.*
2. *It inverts each receiving "$+1$"-share into a "-1"-share to decrease the collected data.*
3. *In the broadcasting phase, it modifies the collected data of other honest node or sends forged messages.*
4. *It broadcasts or forwards inconsistent data.*
5. *It sends data message with an empty data set $\mathcal{Z}$ instead of a non-empty one.*
6. *It sends data message with a non-empty data set $\mathcal{Z}$ instead of an empty one.*
7. *It changes a cycle value.*

Lemma 5.4 (Sharing)**.** *After sending a set of shares, a dishonest node may affect at most $2k+2$ to the final result.*

Proof. This proof is similar to Lemma 4.4 of Chapter 4. □

Lemma 5.5 (Computing collected data)**.** *After computing the collected data, a dishonest node may affect to the final result by* $4k + 2$.

Proof. This proof is similar to Lemma 4.5 of Chapter 4. □

Corollary 5.2. *A dishonest node that corrupts the collected data to be out of the range* $[-2k - 1, 2k + 1]$ *is detected with certainty.*

Proof. This proof is similar to Corollary 4.2 of Chapter 4. □

Lemma 5.6. *In the broadcasting phase, if an honest node* n *broadcasts its collected data* c_n *then all other honest nodes will eventually receive that value.*

Proof. Every honest neighbor of n receives directly the value c_n after the first cycle. For the nodes at distance ≥ 2 from n: by Theorem 5.7 node n stops sending messages in the broadcasting phase: (i) at cycle $i = 1$ if it connects directly to $N - 1$ nodes; or (ii) after $\max\{Diam(G), N - 1 - D\}$ cycles end. Because the network is honest with the presence of D dishonest nodes, the upper bound of honest path's length is $N - 1 - D$. This ensure all honest nodes (even the farthest honest ones from n) receive at least one correct version of c_n. □

Corollary 5.3. *If the set* $\mathcal{C}_n[s]$ *of an honest node* n *is singleton then the single element is a correct collected data of the source node* s.

Proof. By Lemma 5.6, the set $\mathcal{C}_n[s]$ always contains a correct value of source s. This yields the desired result. □

Lemma 5.7 (Broadcasting collected data)**.** *There exists a public verification scheme that detects a dishonest node misbehaving in the broadcasting phase.*

Proof. By Definition 5.1, dishonest nodes can do the following misbehaviors in the broadcasting phase:

1. *Broadcast or forward inconsistent collected data*: the proof is showed in Lemma 4.6 of Chapter 4.

2. *Send many forged collected data*: In one cycle of the broadcasting phase, a node only receives from one direct friend one data message and one acknowledgement messages. Thus, if a dishonest node creates and sends many forged (data or acknowledgement) messages to one honest node, only one of them is accepted, and the remaining messages are dropped at the site of honest node.

3. *Send data message with empty data set* $\mathcal{Z}$ *instead of non-empty one*: In this attack, the coalition receives a non-empty set $\mathcal{Z}$ at cycle i but forwards empty set to its honest neighbors at cycle $i + 1$. This intuition of the dishonest nodes is to prevent honest nodes from continuing following the broadcasting phase of the protocol. (Since if an honest node has received all empty sets $\mathcal{Z}$ from all its neighbors counting from cycle 1 to cycle i, it stops the broadcasting phase and does the verification process.) However, as showed in Theorem 5.7, each honest node still progresses in the system since it receives messages from other honest nodes passed by honest paths which contain non-empty data sets $\mathcal{Z}$. Thus, it is ensured to progress to cycle $i + 1$ and to receive the correct data of all other honest nodes. Moreover, by doing the public verification like cases 1–2 above, the honest nodes can detect this misbehavior.

4. *Send data message with non-empty data set $\mathcal{Z}$ instead of empty one*: This intuition of dishonest nodes is to prevent honest nodes from stopping broadcasting phase. But we see that this case may not affect the accuracy of the honest node's outcome due to the receipt of correct data passed by honest paths. A node may do some additional cycles in this phase, and in the worst case, it does $N-1$ cycles ($N-1$ is the maximal path length in the network) and receive more duplication information. Note that it can detect this misbehavior later by doing verification process like the first case.

5. *Change a cycle value*: A dishonest node u that is currently at cycle i sends a (data or acknowledgement) message containing a cycle value $j \neq i$ to an honest node v, in order to perturb v's progress. However, this attack is detected with certainty because: (i) if $j < i$: node v drops this message as it already finished cycle j (and received all messages of cycle j from u); (ii) if $j > i+1$: by Theorem 5.2, a dishonest node is detected with certainty; and (iii) if $j = i+1$: node v accepts a message of the cycle $j = i+1$ from u if v is executing at step (B4) of the cycle i (by Lemma 5.2) and it already obtained the message (ACK,u,i) from u. (This case does not affect the progress of node u.) Otherwise, u will be detected as dishonest.

□

Theorem 5.8 (Accuracy)**.** *Every dishonest node may affect the expected final result up to* $6k+4$.

Proof. By Lemmas 5.4-5.7, each dishonest node affects to the final outcome at most $2k+2+4k+2 = 6k+4$. Thus, $\sigma = \frac{6k+4}{2N} = \frac{3k+2}{N}$.

Consider a function $\xi(k) = \frac{3k+2}{N}$. With $k \leq \lfloor(-1+\sqrt{3N+1})/3\rfloor$ then $\xi(k) \leq 1/k$. It means $\xi(k)$ decreases as $1/k$; thus, it is negligible. By Definition 4.2, the protocol is accurate. □

Asymptotic complexity. We here examine the complexities of the protocol when deployed for the graphs of family $\mathcal{G}_2$ of size N with the presence of D dishonest nodes.

Proposition 5.4 (Spatial complexity)**.** *The total space each node n must hold is* $\mathcal{O}(k+d_nN(N+D))$.

Proof. Each node n needs to maintain the set of $2k+1$ consumers, the set of $2k+1$ producers, the list of d_n direct neighbors to contact, the set $\mathcal{Z}_n$ to store pairs of identity and collected data for friends, the set $\mathcal{C}_n$ of possible collected data of other nodes, the routing table $\mathcal{T}_n$, and the set h_n to store the final choosing collected data $(N-1)$.

For $\mathcal{Z}_n$: in cycle $i = 1$, it receives one value from a direct neighbor u, and this information is then stored in $\mathcal{Z}_n[w]$ for each direct neighbor $w \neq u$ which takes space of size $d_n - 1$. With d_n direct neighbors, this requires the space of size $(d_n-1)d_n$. At the worst case scenario, a node stops transmitting messages (in the broadcasting phase) after $\max\{Diam(G), N-1-D\}$ cycles end. In each cycle $2 \leq i \leq \max\{Diam(G), N-1-D\}$, n sends at most $N-2$ values for one neighbor u and also receives the same amount of values from u. However, with the existence of D dishonest nodes, it could receive more D different values of a certain node s. Therefore, it needs $((N-2)+(D+1))d_n$ units of storage. This implies, the total space for storing $\mathcal{Z}_n$ in the protocol is $\max\{(d_n-1)d_n, (N-1+D)d_n\} = (N-1+D)d_n$.

Similarly, for $\mathcal{T}_n$: in cycle 1, it stores d_n values. In each cycle $2 \leq i \leq \max\{Diam(G), N-1-D\}$, it requires at most $(N-2+D+1)d_n$ units of storage. As $Diam(G) \leq N-1$, and $N-1-D \leq N-1$, we consider a nodes does $N-1$ cycles in the worst case. Hence, the space of $\mathcal{T}_n$ is $\mathcal{O}(d_n + (N-2)(N-1+D)d_n) = \mathcal{O}(d_nN(N+D))$.

For $\mathcal{C}_n$: in the worst case, for each honest source's data, a dishonest coalition can forward to n at most D different values, and n also receives one correct value from the honest path. Hence, the maximum size of $\mathcal{C}_n$ is $(N-1)(D+1)$.

Consequently, the total space each node must hold in the worst case is $\mathcal{O}(k)+\mathcal{O}(d_n)+\mathcal{O}((N-1+D)d_n)+\mathcal{O}(d_nN(N+D))+\mathcal{O}((N-1).(D+1))+\mathcal{O}(N-1)=\mathcal{O}(k+d_nN(N+D))$. □

Proposition 5.5 (Message complexity)**.** *Each node n sends $\mathcal{O}(k+Nd_n)$ messages.*

Proof. In the sharing phase, a node n sends $2k+1$ messages to its consumers. In the broadcasting phase, it sends $2d_n$ data and acknowledgement messages to all of its direct neighbors. There are at most $\max\{Diam(G), N-1-D\} \leq N-1$ cycles (see Theorem 5.7 for more details of this bound). Accordingly, the message complexity is $\mathcal{O}(2k+1)+\mathcal{O}(2d_n.N)=\mathcal{O}(k+Nd_n)$. □

Proposition 5.6 (Time complexity)**.** *Assume time evolves in rounds, i.e., each message transmission incurs a delay of at most one round. Then the protocol operates in $\mathcal{O}(k+N.\max_n\{d_n\})$ rounds.*

Proof. The sharing phase operates in $2k+1$ rounds. In the broadcasting phase, a node n sends d_n messages to direct neighbors, and all the nodes broadcast their collected data in parallel, so this activity takes at most $\max_n\{d_n\}$ rounds. Node has to repeat at most $\max\{Diam(G), N-1-D\} \leq N-1$ cycles. Therefore, the time complexity is $\mathcal{O}(2k+1)+\mathcal{O}(\max_n\{d_n\})+\mathcal{O}(\max_n\{2d_n\}.N)=\mathcal{O}(k+N.\max_n\{d_n\})$. □

The necessary and sufficient conditions with dishonest nodes

We analyze the necessary and sufficient conditions for our protocol to be deployed correctly for graphs of family $\mathcal{G}_2$.

Theorem 5.9. *The properties of $\mathcal{G}_2$ are the necessary and sufficient conditions for the polling protocol to be deployed correctly in the system with the presence of dishonest nodes. It should be noted that the family of graphs $\mathcal{G}_2$ is similar to the one in Chapter 4.*

Proof. The proof is similar to the one of Theorem 4.7 of Chapter 4. □

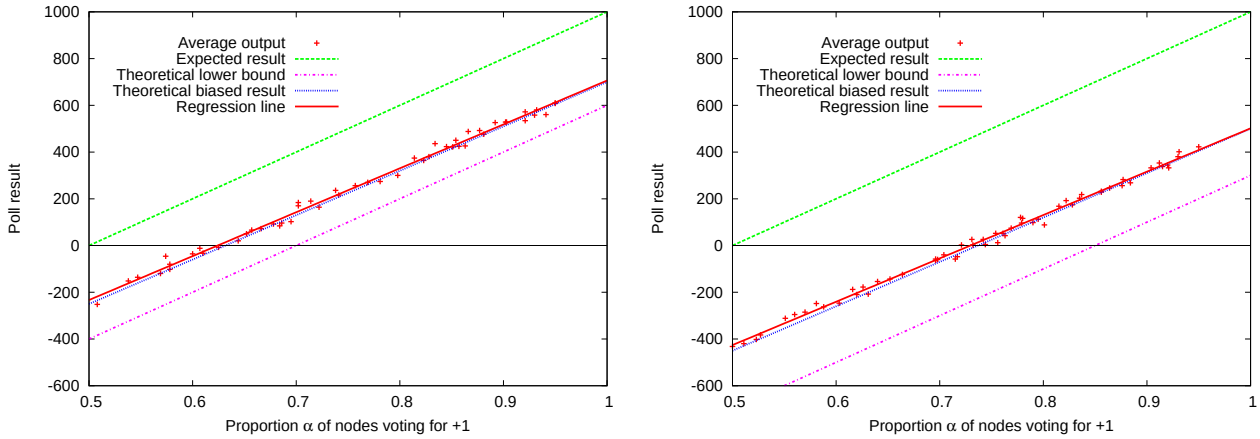
5.4 Experimental evaluation

We validate our solution with a performance evaluation by observing the difference between experiment output and the theoretical one.

Experimental setup. In the experiments, we use UDP and asynchrony in message exchanging without crash or message loss. We implement protocol by using framework MaDKit[12] to demonstrate the communication amongst nodes and facilitate the development and testing of the applications. We conduct the experiments on the local machine with the following configuration: Intel Core 2 Quad CPU Q9550 @ 2.83GHz × 4 and Ubuntu OS 11.10.

Like previous chapter, we study the experiment with the existence of dishonest nodes, and examine whether the real impact from the dishonest coalition to the outcome is within the analytical bound or not. Namely, we consider the worst case for the system in the sense that the dishonest nodes try to misbehave protocol by expressing the attack to decrease the final result at most without being detected: each dishonest node always sends $2k+1$ shares of value

[12] http://www.madkit.org/



(a) $k = 1$ (b) $k = 2$

Figure 5.3: Experiment with $N = 1000$, $D = 50$ to check the accuracy of protocol.

"-1" in the sharing phase, and converts all receiving shares of "1" into ones of "-1". Thus, each dishonest node affects the final result at most $2k + 2$ in the sharing phase (if its desired vote is "1" and it sends $2k + 1$ shares of "-1") and $2(2k + 1) = 4k + 2$ in the broadcasting phase (if $2k + 1$ receiving shares are "1" and all are converted to "-1"). In other words, total impact will be up to $6k + 4$. If we denote by α number of nodes voting "$+1$", then the expected result will be $\alpha N - (1 - \alpha)N = (2\alpha - 1)N$. So theoretically, the biased final outcome should be within the interval $[(2\alpha - 1)N - (6k + 4)D; (2\alpha - 1)N]$.

Results. We examine the experiments for the network with $N = 1000$, $D = 50$ in two subcases corresponding to two different values of privacy parameter $k = 1$ and $k = 2$. Fig. 5.3a and 5.3b depict our results for these two test cases. In each test, we compute the average output over all nodes and represented it as a point in the figures. We see that the experimental result is certainly within two expected theoretical bounds: upper bound (thick-dashed line) and lower bound (dot-dashed line). The experiment result never touches the expected line as the dishonest node always sends $2k + 1$ shares of "-1" and converts all receiving shares of "$+1$" to "-1" and this decreases the final output and makes it less than the expected outcome. Moreover, the experiment result does not reach the theoretical lower bound either, i.e., the average impact is less than $6k + 4$.

Furthermore, similar to the analysis in previous chapter, with D dishonest nodes, the average biased outcome is $(2\alpha - 1)N - (4k + 2\alpha + 2)D$. In Fig. 5.3, we present this average value in a thin-dotted line. We try to fit our data points with a regression line $a(2\alpha - 1) - b(4k + 2\alpha + 2)$ and get these results (depicted as a solid line in Fig. 5.3): for $k = 1$: $a = 986$ and $b = 47$, and for $k = 2$: $a = 974$ and $b = 47$. These parameters are quite accurate compared to conditions of network with $N = 1000$ and $D = 50$.

In Fig. 5.3, we see that the impact of dishonest nodes for $k = 2$ is greater than for $k = 1$. This result is reasonable since we know that the higher value k is, the higher privacy can be hold but the higher impact dishonest nodes can enforce, and hence, the worse the final outcome is. Besides, all nodes output the correct result (i.e., the final result is greater than 0) for $k = 1$ when $\alpha \geq 0.63$, and for $k = 2$ when $\alpha \geq 0.74$. It means dishonest nodes confuse the majority of nodes for $k = 1$ when $\alpha < 0.63$, and for $k = 2$ when $\alpha < 0.74$.

5.5 Summary and discussion

We presented a design of an asynchronous distributed polling protocol that rely on current state of social graphs. This protocol is the improvement of the one showed in last chapter since it runs in asynchronous network model. We also described the verification process to dissuade misbehaviors, where each user does not need any knowledge of shortest path lengths. Both (synchronous and asynchronous) protocols require verification procedures to ensure the safety and security, and their communication cost are super-linear in N. Next chapter, we will present an asynchronous polling protocol which overcomes these shortcomings: it does not require any verification procedures and contains a method for efficiently broadcasting message.

Chapter 6

Polling Protocol with Efficient Communication

Contents

6.1 Polling model 84
6.1.1 Social graph model 84
6.1.2 Secret sharing based graphs 85
6.2 Polling protocol 85
6.3 Correctness and Complexity Analysis 88
6.3.1 Protocol and graph without dishonest nodes 88
6.3.2 Protocol and graph with dishonest nodes 89
6.4 Crash and message loss analysis 96
6.5 Particular graphs 97
6.6 Summary and discussion 98

Previously, we have described two simple decentralized polling protocols that run in synchronous and asynchronous network models and rely on the current state of social graphs. In these protocols, the dishonest participants are tolerated but their misbehaviors are detected with positive probability by using verification procedures. In addition, the communication cost are super-linear in N (they are $\mathcal{O}(N^2)$ with the presence of dishonest nodes).

In this chapter, we present EPol, an asynchronous decentralized polling protocol that relies on the current state of social graphs, not based on any verification procedures, and the communication and spatial complexities are close to be linear. More explicitly, we define one family of social graphs that satisfy what we call the m-broadcasting property (where m is less than or equal to the minimum node degree) and show their structures enable low communication cost and constitute necessary and sufficient condition to ensure vote privacy and limit the impact of dishonest users on the accuracy of the polling output. Different from previous algorithms in Chapters 4 and 5, in this protocol, with a privacy parameter k, each user can vote with $2i + 1$ shares where $i \in \{0, 1, ..., k\}$. We then analyze the probabilities for disclosing votes with certainty and greedy (we explain more later in Section 6.3.2).

Since the data sent by one node may be corrupted by intermediate dishonest nodes, an honest node may receive distinct values of the same data. As stated in [82, 83], the honest node may decide on the arbitrary data (aka. local tally) sent by dishonest nodes and thus it might be

incorrect. In contrast, we ensure each node can decide the most represented value to obtain correct data of other nodes.

Moreover, most of the previous works [24, 25, 65, 82, 83] assume the existence of reliable communication among nodes. However, nodes communicate by UDP which may suffer message loss on the communication channels or nodes' crash. In this chapter, we analyze the affect of these factors on accuracy and termination of the protocol by considering the impact on the final outcome and the probability of a node failing to decide and compute the final result. It is also noted that DPol [82, 83] investigated the effect only from crash, and not from message loss.

The remaining of chapter is organized as follows. Section 6.1 describes the ingredients of our polling model. Section 6.2 presents our polling protocol and Section 6.3 analyzes its correctness with and without presence of dishonest nodes. Section 6.4 discusses the impact of crash and message loss on accuracy and termination of the protocol. Section 6.5 shows examples of our social graph structures. We summarize and conclude this chapter in Section 6.6.

6.1 Polling model

In this section, we define the social graph models and demonstrate the description of our family of graphs. We consider the same user behaviors given in Section 5.1.1 of Chapter 5 (and we omit it here).

6.1.1 Social graph model

We use the terms and notations of graphs given in Chapter 3 to describe social graph.

In the network, consider the broadcasting operation initiated by a single node, called *source*. The source has a data and wants to disseminate it to all nodes in the network. In the naive approach, upon receiving a message from a neighbor, a node stores the data then forwards to all other neighbors. Despite the use of richer social graph structures, one node can receive/send so many duplicated messages (which may be passed by many paths) from/to other nodes. This leads to flooding the local storage. To overcome this problem, inspired from [166], we propose a method for efficiently broadcasting messages by using the concept that we call the *m-broadcasting property*.

Definition 6.1 (m-broadcasting property)**.** *A graph satisfies the* m-broadcasting property *for a positive integer m such that $1 \leq m \leq d_{min}$, where d_{min} is the minimum node degree of the network, if for each source node, there exists a topological ordering of the nodes in the graph such that every node either connects directly to the source or to some m nodes preceding it in the ordering w.r.t. the source.*

Accordingly, instead of accepting all messages originating from a source, a node receives and stores only m ones passed by ordered paths.

We denote by $\beta_n(s)$ a number of neighbors of n preceding it in the ordering w.r.t. source s.[13] (We sometimes omit the mentioned source where no confusion arises.) Section 6.5 describes some examples of graphs satisfying this condition. Just to demonstrate one typical example, for a graph satisfied 3-broadcasting property in Fig.6.1a (where each node's order is represented in the parentheses) and node A does the broadcast operation for its data. In the naive approach, node F can receive all messages passed by all the paths between F and A. However, by using 3-broadcasting property, F just receives three messages from neighbors E, B and D.

[13]The list of neighbors is determined by a preprocessing step before the polling process.

The construction of a graph satisfying the m-broadcasting property from a general graph is beyond the scope of this work.

6.1.2 Secret sharing based graphs

In this part, we present the family of graphs including the ideal case (network without dishonest nodes) and normal case (network with the presence of dishonest nodes).

Like previous chapters, we use a predefined parameter $k \in \mathbb{N}$ and $k \leq \lfloor(-1+\sqrt{3N+1})/3\rfloor$ [82, 83] and a parameter $m \in \mathbb{N}$ to describe the features of our social graphs. Let $G = (V, E)$ be a social graph with the following properties:

Property 6.1 (P_{g_1}). *$d_n \geq 2k+1$, $|\mathcal{S}_n| = 2i+1$ and $|\mathcal{R}_n| \leq 2k+1$ where $i \in \{0, 1, ..., k\}$, for every $n \in V$.*

Property 6.2 (P_{g_2}). *G satisfies the m-broadcasting property.*

Property 6.3 (P_{g_3}). *For a source node, each other node has less than $m/2$ dishonest neighbors preceding it in the ordering (w.r.t. source node).*

According to Property P_{g_1}, a set of consumers and a set of producers of one node have the size of *at most* $2k+1$ and might not be disjoint. This condition distinguishes our graph family from other structures discussed in [65,82,83] and is more flexible than graph families in Chapters 4 and 5 since they all consider the restricted condition where each node has *exactly* $2k+1$ consumers and producers. In addition, it also differs from [24, 25] which do not give any condition on the upper bound of the number of producers (that each node should have). Thus, a dishonest node can send arbitrary summing data to others and the accuracy of global outcome is easily affected. Property P_{g_2} enables us to reduce the communication cost in the system. It is also noted that this condition implicitly implies the condition that G is an honest graph mentioned in previous chapters (i.e., for every honest nodes u, v, there exists a path between u and v containing only intermediate honest nodes). Property P_{g_3} ensures each honest node always obtains one correct version of data from other honest ones. Property P_{g_3} also enables us to limit the size of dishonest users, that is $D \leq \frac{m-1}{2} Diam(G)$ (presented in Theorem 6.8).

From these properties, we characterize two families of graphs:

(i) $\mathcal{G}_1 = \{G \mid \mathcal{D}(G) = \varnothing$ and G satisfies $P_{g_1}, P_{g_2}\}$.

(ii) $\mathcal{G}_2 = \{G \mid \mathcal{D}(G) \neq \varnothing$ and G satisfies P_{g_1}, P_{g_2} and $P_{g_3}\}$.

6.2 Polling protocol

In this section, we present our polling protocol, *EPol*, for the network without crash and message loss. EPol is composed of the following phases:

Sharing. In this phase, each node n contributes its opinion by sending a set of shares expressing its vote $v_n \in \{-1, 1\}$ to its consumers. We inspired the sharing scheme proposed in [61] to generate shares. Namely, first n chooses *randomly* a value i such that $i \in \{0, 1, ..., k\}$. This value i is not known by other nodes. Then it generates $2i+1$ shares $\mathcal{P}_n = \{p_1, p_2, ..., p_{2i+1}\}$, where $p_j \in \{-1, 1\}$ and $1 \leq j \leq 2i+1$, including: $i+1$ shares of value v_n, and i shares of

Algorithm 3: POLLING ALGORITHM AT NODE $n \in \{0, 1, ..., N-1\}$

Input:
v_n: A vote of node n, value in $\{-1, 1\}$
k : privacy parameter
m : positive integer where $1 \leq m \leq d_{min}$

Variables:
c_n: collected data, $c_n = 0$
$\mathcal{C}_n$: set of possible collected data
$\mathcal{C}_n[\{0, 1, ..., N-1\} \to \varnothing]$
h_n: set of final deciding collected data
$h_n[\{0, 1, ..., N-1\} \to \perp]$

Output: result

Polling Algorithm

1 `Sharing`$(v_n, \mathcal{S}_n, i)$
2 `Broadcasting`(c_n)
3 `Aggregating`$()$

Procedure `Sharing`$(v_n, \mathcal{S}_n, i)$

4 $\mathcal{P}_n \leftarrow \{v_n\}$
5 **for** $j \leftarrow 1$ **to** i **do**
6 $\quad \mathcal{P}_n \leftarrow \mathcal{P}_n \cup \{v_n\} \cup \{-v_n\}$
7 $\mu_n \leftarrow_{rand} \mathcal{P}_n$
8 **for** $j \leftarrow 0$ **to** $2i$ **do**
9 $\quad$ send (SHARE, $\mu_n[j]$) to $\mathcal{S}_n[j]$
10 $count \leftarrow 0$
11 **while** ($count < |\mathcal{R}_n|$) **do**
12 $\quad$ **upon event** (receive (SHARE, p) from neighbor r in the first time) **do**
13 $\quad\quad$ **if** ($r \in \mathcal{R}_n$ **and** $p \in \{-1, 1\}$) **then**
14 $\quad\quad\quad c_n \leftarrow c_n + p$
15 $\quad\quad\quad count \leftarrow count + 1$

Procedure `Broadcasting`(c_n)

16 **foreach** ($r \in \Gamma(n)$) **do**
17 $\quad$ send (DATA, n, c_n) to r
18 $count \leftarrow 0$
19 **while** ($count < N - 1$) **do**
20 $\quad$ **upon event** (receive message (DATA, s, c_s) from direct neighbor r preceding n in the ordering w.r.t. source s) **do**
21 $\quad\quad$ **if** ($r = s$) **then**
22 $\quad\quad\quad h_n[s] \leftarrow c_s$
23 $\quad\quad\quad count \leftarrow count + 1$
24 $\quad\quad\quad$ Forward (DATA, s, c_s) to other friends succeeding n in the ordering w.r.t. source s
25 $\quad\quad$ **else if** ($s \notin \Gamma(n)$) **then**
26 $\quad\quad\quad \mathcal{C}_n[s] \leftarrow \mathcal{C}_n[s] \cup \{c_s\}$
27 $\quad\quad\quad$ **if** ($|\mathcal{C}_n[s]| = m$) **then**
28 $\quad\quad\quad\quad h_n[s] \leftarrow$ `Decide`$(\mathcal{C}_n[s])$
29 $\quad\quad\quad\quad count \leftarrow count + 1$
30 $\quad\quad\quad\quad$ send (DATA, s, $h_n[s]$) to other $d_n - \beta_n(s)$ friends succeeding n in the ordering w.r.t. source s

Function `Decide`$(\mathcal{Z})$

31 **return** the most represented value in $\mathcal{Z}$

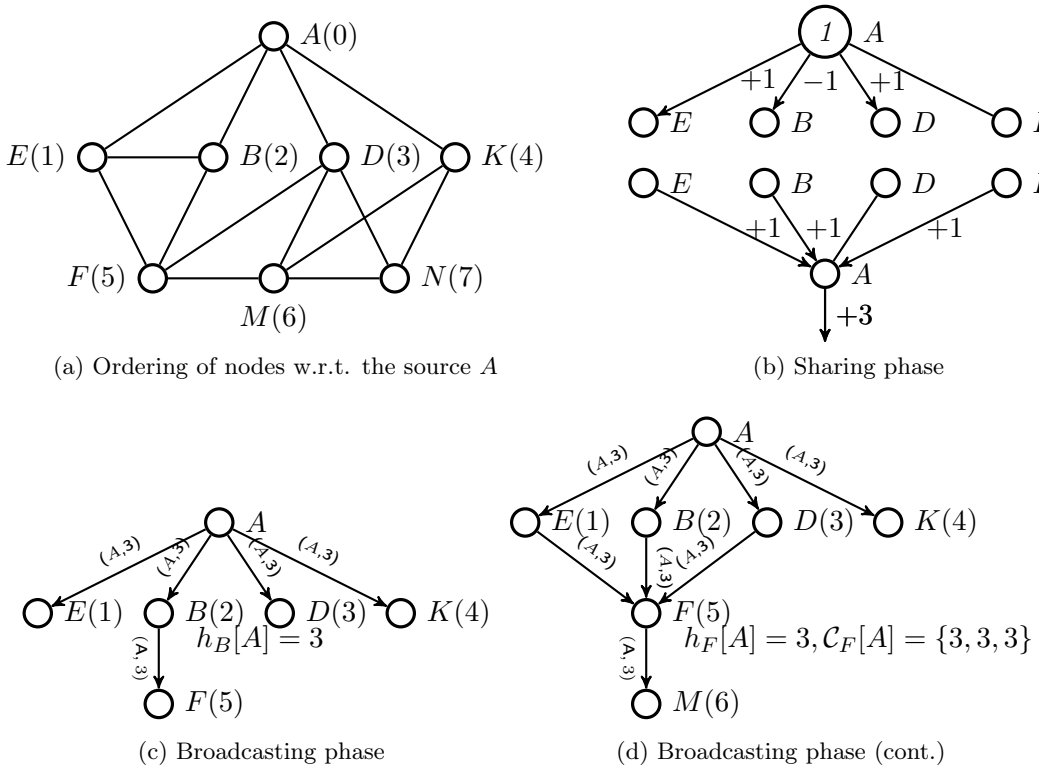
Procedure `Aggregating`$()$

32 result $\leftarrow 0$
33 **for** $s \leftarrow 0$ **to** $N - 1$ **do**
34 $\quad$ **if** ($s \neq n$) **then**
35 $\quad\quad$ result $\leftarrow$ result $+ h_n[s]$
36 $\quad$ **else** result $\leftarrow$ result $+ c_n$

opposite v_n's value. The intuition of this creation is to regenerate the vote v_n when the shares are summed. Later it randomly generates a permutation μ_n of $\mathcal{P}_n$, and sends shares to $2i + 1$ consumers. Lines 4–9 in Algorithm 3 describe this activity. Node also receives $|\mathcal{R}_n|$ shares from its producers. Note that $\mathcal{S}_n$ and $\mathcal{R}_n$ might not be disjoint.[14] After each node collects shares from its producers, and sums into *collected data* c_n (lines 10–15 in Algorithm 3), this phase is over. It is also noted that because the votes and their generating shares belong to the set $\{-1, +1\}$, nodes cannot distinguish between a vote and a share. Hence, if a node opts a value $i = 0$ and generates only $2i+1 = 1$ share, the dishonest consumer receiving a message from that node could not distinguish if such share was generated as a single one or it is one among many generated shares of that node.

Figure 6.1 illustrates an example of the protocol for $i = k = 1$. Figure 6.1a presents the network and the ordering of nodes w.r.t. source A in the parentheses. Figure 6.1b depicts the sharing phase at node A. Node A would like to vote $+1$, thus, it generates a set of $2i + 1 = 3$ shares $\{+1, -1, +1\}$. Node A collects the shares from its producers and computes the collected data $c_A = 3$.

[14] This distinguishes our protocol from approaches in [24, 25, 65, 82, 83]. The set of consumers and producers in these approaches are separated for each of size $2k + 1$.



(a) Ordering of nodes w.r.t. the source A

(b) Sharing phase

(c) Broadcasting phase

(d) Broadcasting phase (cont.)

Figure 6.1: Polling algorithm for $i = k = 1$ and $m = 3$.

Broadcasting. In this phase, each node needs to disseminate its collected data to all other nodes in such a way that each other node eventually obtains that correct data. In the naive approach, upon receiving a message from the neighbor, a node stores the data then forwards it on every other edge. Despite the use of richer social graph structures, and with the presence of dishonest nodes which can corrupt data, one node can receive/send so many duplicated messages (which may be passed by many paths) from/to other nodes. This leads to flooding the local storage. As motivated in Section 6.1.2, we propose a method for efficiently broadcasting messages by using the m-broadcasting property. For a graph satisfying the m-broadcasting property, each node n first sends its collected data to all neighbors (lines 16–17). Then, upon receipt of the message containing the collected data of source s from a neighbor r preceding it in the ordering (w.r.t. source s), node n executes one of the following activities:

- $r = s$: It decides on the data of source s by storing the received value c_s in $h_n[s]$. When the value $h_n[s]$ is assigned, it is further forwarded to all $d_n - \beta_n(s)$ nodes succeeding it in the ordering (lines 21–24).

  In Fig. 6.1c, after node A broadcasts its data, node B receives, stores this data in $h_B[A]$, and forwards it to F.

- $r \neq s$: To avoid the case where the value $h_n[s]$ might be computed (and broadcast) twice for direct neighbor s, node n only considers the case $r \neq s$ and $s \notin \Gamma(n)$. If that condition is satisfied, it adds the value c_s to the multiset $\mathcal{C}_n[s]$ of possible collected data for s (line 26). When node n has received the expected number m of possible collected data for a given source s, it decides on the collected data by choosing the most represented value

in $\mathcal{C}_n[s]$ and puts it in $h_n[s]$. (Since the decision is based on the most represented value, instead of waiting for receiving all m forwarded data, node n can decide the source's data upon receipt of more than $m/2$ identical data.) Node n then further forwards the data to all $d_n - \beta_n(s)$ nodes succeeding it (lines 27–30).

Fig. 6.1d depicts node F receives messages from its neighbors about the data of source A. It has four friends, but receives only $m = 3$ messages from preceding neighbors E, B and D. As all values in $\mathcal{C}_F[A]$ are 3, node F decides that value as the collected data of A and stores it in $h_F[A]$. It then forwards that data to its succeeding node M.

When a node decides the collected data of s and has no succeeding friend, the value $h_n[s]$ is no longer forwarded. This phase is complete if a node decides on the collected data of all other ones in the network.

Aggregating. The final result is obtained at each node by simply summing all deciding collected data in the set h_n and its own collected data: $\mathsf{result} = c_n + \sum_{i \neq n} h_n[i]$ (lines 32–36).

6.3 Correctness and Complexity Analysis

In this section, we present the correctness and complexity analysis of our protocol when deployed on the graphs of $\mathcal{G}_1$ and $\mathcal{G}_2$ without and with the presence of dishonest nodes.

6.3.1 Protocol and graph without dishonest nodes

We first analyze the correctness (including accuracy and termination) of our protocol for the graphs of $\mathcal{G}_1$ in which all participants are honest in Theorem 6.1. Then we analyze the spatial, message and time complexities in Propositions 6.1–6.2.

Correctness

We show the correctness of the protocol by proving its accuracy and termination properties in the following theorem.

Theorem 6.1 (Correctness)**.** *Consider a polling system of size N with only honest nodes where each node n expresses a vote v_n. The polling algorithm is guaranteed to eventually terminate and each node outputs $\sum_{n=0}^{N-1} v_n$.*

Proof (Accuracy). In the sharing phase, each node n sends a set of shares $\mathcal{P}_n = \{p_{n_1}, p_{n_2}, ..., p_{n_{2i+1}}\}$ to its consumers where $\sum p_{n_j} = (i+1).v_n + i.(-v_n) = v_n$, and also receives a set of shares $\{p'_{n_1}, p'_{n_2}, ..., p'_{n_l}\}$ from its producers ($l = |\mathcal{R}_n|$) to obtain a collected data of value $c_n = \sum_{j=1}^{l} p'_{n_j}$. With the assumption there is no dishonest node and without crash or message loss, each message from the source successfully reaches the destination, and thus the set of all sending shares of all nodes will be exactly coincided with the set of all receiving shares of all nodes, namely:

$$\bigcup_V \{p_{n_1}, p_{n_2}, ..., p_{n_{2i+1}}\} = \bigcup_V \{p'_{n_1}, p'_{n_2}, ..., p'_{n_l}\}$$

In the broadcasting phase, each node n broadcasts its collected data to their neighbors, then they do honestly forward that value to neighbors of n's neighbors (succeeding it in the ordering w.r.t. source n) and so on. The messages are finally received by all direct and indirect

neighbors. Node n builds an array h_n that contains all collected data of all nodes in the system. Consequently, the final computation gives us the value:

$$\mathsf{result} = \sum_{\substack{0 \leq j < N \\ j \neq n}} h_n[j] + c_n = \sum_{j=0}^{N-1} c_j = \sum_{j=0}^{N-1} \sum_{t=1}^{l} p'_{j_t} = \sum_{j=0}^{N-1} \sum_{t=1}^{2i+1} p_{j_t} = \sum_{j=0}^{N-1} v_j$$

Proof (Termination). In the sharing phase, each node has to receive a finite number ($|\mathcal{R}|$) of messages. In the broadcasting phase, for a source s, each direct neighbor of s receives only one message, and each other indirect neighbor receives m messages. Since every node sends the required number of messages and they are eventually arrived to destination, each phase completes. As the protocol has a limited number of phases, it is ensured to eventually terminate. □

Asymptotic complexity. We here examine the complexities of the protocol in Propositions 6.1–6.2.

Proposition 6.1 (Spatial complexity)**.** *The total space each node must hold is* $\mathcal{O}(k + m.N)$.

Proof. Each node n maintains a set of producers and consumers (at most $2(2k + 1)$), a list of d_n direct neighbors, a set of N identities of nodes in the systems, a set of m possible collected data for each of source s, a set h_n to store the deciding collected data. Therefore, the spatial complexity is $\mathcal{O}(k) + \mathcal{O}(d_n) + \mathcal{O}(N) + \mathcal{O}(m.(N-1)) + \mathcal{O}(N-1) = \mathcal{O}(k + m.N)$. □

Proposition 6.2 (Message complexity)**.** *The average number of messages sent by a node* n *is* $\mathcal{O}(k + N.(d_n - m))$.

Proof. In the sharing phase, node n sends at most $2k + 1$ messages. In the broadcasting phase, it sends d_n messages containing its collected data, and forwards at most $d_n - m$ messages when receiving collected data of each source s from its neighbors. Accordingly, the message complexity is $\mathcal{O}(2k + 1) + \mathcal{O}(d_n) + \mathcal{O}((N-1).(d_n - m)) = \mathcal{O}(k + N.(d_n - m))$. □

6.3.2 Protocol and graph with dishonest nodes

In this section, we study the impact of dishonest nodes on privacy and accuracy when EPol is deployed on the graphs of $\mathcal{G}_2$ in the worst and average cases.

Privacy

We denote by γ_i the proportion of nodes voting with $2i + 1$ shares in the sharing phase, where $0 \leq i \leq k$ and $\sum_{i=0}^{k} \gamma_i = 1$. We consider two cases for disclosing a nodes' vote as follows.

Vote disclosing with certainty. We discuss the case when the vote of a given node may be disclosed with certainty in the following theorem.

Theorem 6.2 (Certain Privacy)**.** *Assume a coalition of* D *dishonest nodes knows the number of shares sent by a node. The probability* P_{ce} *this coalition reveals correctly with certainty a vote of the honest node voting with* $2i + 1$ *shares* $(0 \leq i \leq k)$ *is at most* $\gamma_i \left(\frac{D}{N}\right)^{i+1}$.

Proof. The coalition reveals vote v of a node if and only if $i+1$ consumers receiving $i+1$ identical shares of value v belong to the dishonest coalition. There are a proportion γ_i of these nodes. Thus, that event occurs with probability $P_{ce} = \gamma_i \binom{D}{i+1} / \binom{N}{i+1} \leq \gamma_i \left(\frac{D}{N}\right)^{i+1}$. □

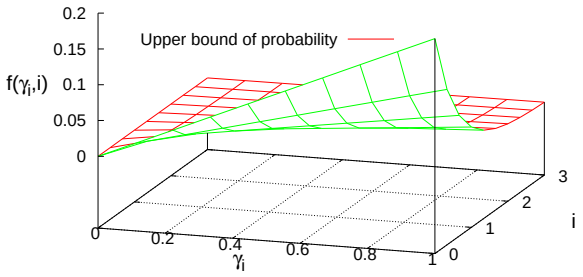


Figure 6.2: Probability to disclose a node vote with certainty.

Corollary 6.1. *If all nodes send $2k+1$ shares in the sharing phase, then the probability that a coalition of D dishonest nodes reveals correctly with certainty an honest node's vote is at most $\left(\frac{D}{N}\right)^{k+1}$.*

Proof. The claim is followed by the result of Theorem 6.2. □

We plot the bound of P_{ce} as a function $f(\gamma_i, i) = \gamma_i\left(\frac{D}{N}\right)^{i+1}$ in Fig. 6.2 for $k = 3$, $N = 100$ and $D = 20$. We see that P_{ce} increases with the increase of γ_i and the decrease of i. Thus, we get the maximum privacy when all nodes generate $2k+1$ shares, and the minimum privacy when all nodes generate only one share.

If the poll outcome is N (resp. $-N$), it implies all nodes vote for "+1" (resp. "−1") and they all are disclosed. Moreover, w.l.o.g., assume each dishonest node always votes for "−1". Thus, if the result is $N - 2D$ (resp. $-N$) then it implies all honest nodes vote for "+1" (resp. "−1"). Without considering this case, i.e., all honest nodes do not vote for the same option, Theorem 6.3 gives us the maximum number of votes the dishonest coalition could discover.

Theorem 6.3. *If all honest nodes do not vote for the same option, a coalition of D dishonest nodes can reveal at most $2D$ votes of honest nodes.*

Proof. A consumer receives on average $\sum_i \gamma_i(2i+1)$ shares; hence, the dishonest coalition collects at most $D\sum_i \gamma_i(2i+1)$ shares. Moreover, a vote v of one node voting with $2i+1$ shares is revealed if and only if the coalition obtains $i+1$ identical shares of value v. Thus it recovers at most $\lfloor D.\frac{\sum_i \gamma_i(2i+1)}{\sum_i \gamma_i(i+1)} \rfloor \leq 2D$ votes. □

Vote disclosing without certainty. This part examines the case where the dishonest nodes collude to reveal an honest node's vote without sureness. The coalition decides a node's vote based on the received shares in the sense dishonest nodes can decide the vote after getting some shares or after getting all shares from that node. Thus, they choose one of the following two strategies: (a) Upon receipt of $\rho+1$ identical shares (for some $0 \leq \rho \leq k$) from a given node, they will be considered as its vote; (b) After receiving all shares from a given node, the most represented value of the received shares will be considered as its vote. The former strategy is used by the "greedy" dishonest users who want to reveal rapidly the honest user's vote (even if they have just received one share). The latter one is used by the "non-greedy" dishonest users who are patient and wait for receiving all node's shares before trying to disclose the vote. We

present the probabilities that a coalition of dishonest nodes discloses an honest node's vote for these situations in Theorems 6.4 and 6.5.

Theorem 6.4 (Greedy vote disclosing)**.** *Assume a coalition of D dishonest nodes agrees on the following rule "upon receipt of $\rho+1$ identical shares ($0 \leq \rho \leq k$) from a given node, they will be considered as the node's vote". The probability this coalition reveals correctly a node's vote is $P_{gr}(\rho) = \sum_{i=\rho}^{k} \gamma_i \cdot \binom{D}{\rho+1} \sum_{j=0}^{\rho} \binom{D-\rho-1}{j} / \binom{N}{j+\rho+1}$ and is bounded by $\sum_{i=\rho}^{k} \gamma_i \frac{N+1}{N-D+\rho+2} (\frac{D}{N-D+\rho+1})^{\rho+1}$.*

Proof. The dishonest nodes succeed to discover a node's vote v if that node has sent $2i+1 \geq 2\rho+1$ shares in which $\rho+1$ identical ones representing v and up to ρ shares of value $-v$ were received by the dishonest consumers. In contrast, the coalition's decision is failed if the node has sent more than $2\rho+1$ messages (i.e., at least $2\rho+3$ messages) but the dishonest nodes obtained only $\rho+1$ messages of value $-v$ and up to ρ messages of value v.

The probability a coalition of D dishonest nodes discloses correctly a vote v is: $P_{gr}(\rho) = \sum_{i=\rho}^{k} \gamma_i p(i)$ where $p(i) = \binom{D}{\rho+1} \sum_{j=0}^{\rho} \binom{D-\rho-1}{j} / \binom{N}{j+\rho+1}$.

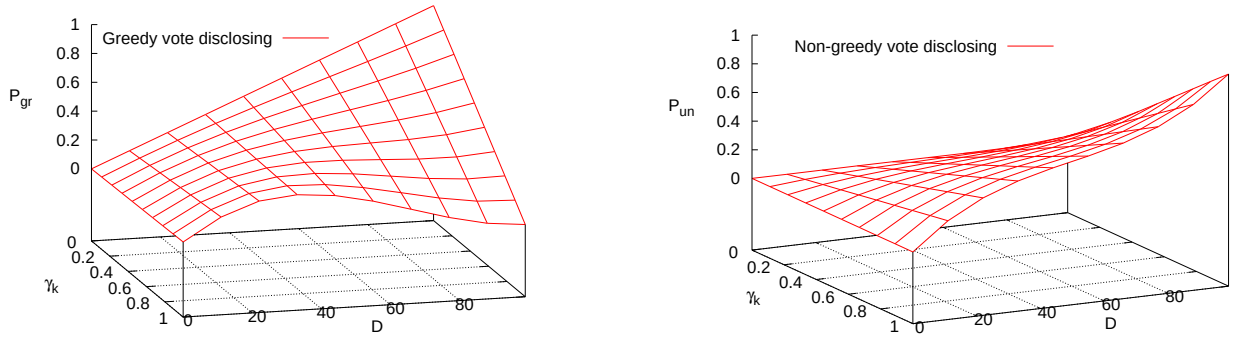
Using this identity: $1/\binom{n}{j} = (n+1) \int_0^1 t^j (1-t)^{n-j} dt$ for some positive n and j, then $p(i)$ is rewritten as follows:

$$
\begin{aligned}
p(i) &= (N+1)\binom{D}{\rho+1} \sum_{j=0}^{\rho} \binom{D-\rho-1}{j} \int_0^1 t^{j+\rho+1}(1-t)^{N-j-\rho-1} dt \\
&= (N+1)\binom{D}{\rho+1} \int_0^1 (1-t)^{N-\rho-1} t^{\rho+1} \left[\sum_{j=0}^{\rho} \binom{D-\rho-1}{j} \left(\frac{t}{1-t}\right)^j \right] dt \\
&\leq (N+1)\binom{D}{\rho+1} \int_0^1 (1-t)^{N-\rho-1} t^{\rho+1} \left[\sum_{j=0}^{D-\rho-1} \binom{D-\rho-1}{j} \left(\frac{t}{1-t}\right)^j \right] dt \\
&= (N+1)\binom{D}{\rho+1} \int_0^1 (1-t)^{N-\rho-1} t^{\rho+1} \left[\left(1+\frac{t}{1-t}\right)^{D-\rho-1} \right] dt \\
&= (N+1)\binom{D}{\rho+1} \int_0^1 t^{\rho+1}(1-t)^{N-D} dt \\
&= \frac{N+1}{N-D+\rho+2}\binom{D}{\rho+1} \Big/ \binom{N-D+\rho+1}{\rho+1} \leq \frac{N+1}{N-D+\rho+2} \left(\frac{D}{N-D+\rho+1}\right)^{\rho+1}.
\end{aligned}
$$

This leads the desired result. □

In Theorem 6.4, a vote v of the honest node is discovered if that node has sent $2i+1 \geq 2\rho+1$ shares in which $\rho+1$ identical ones representing v and up to ρ shares of value $-v$ were received by the dishonest consumers. Moreover, by Theorem 6.4, value P_{gr} increases when γ_i decreases (and i increases) and D increases. For example, with $N = 100$, $k = 1$ (i.e., each node votes with one share or $2k+1 = 3$ shares), $\rho = 0$, we plot the probability P_{gr} as a function of D and γ_k in Fig. 6.3a. As expected, the vote privacy decreases (i.e., P_{gr} increases) when γ_k decreases and D increases.

Theorem 6.5 (Non-greedy vote disclosing)**.** *Assume a coalition of D dishonest nodes agrees on the following rule "the most represented value of the received shares from a given node will be considered as the node's vote". The probability this coalition reveals correctly a node's vote is $P_{un} = \sum_{i=0}^{k} \gamma_i \cdot \sum_{j=1}^{i+1} \sum_{t=0}^{j-1} \binom{D}{j} \binom{D-j}{t} / \binom{N}{j+t}$ and is bounded by $\left(\frac{D}{N} / (1-2\frac{D}{N})\right) \left[1 - \sum_{i=0}^{k} \gamma_i (2\frac{D}{N})^{2i+1}\right]$.*



(a) Greedy (b) Non-greedy

Figure 6.3: Probability to disclose a node vote without certainty.

Proof. The dishonest nodes reveal successfully a vote v of a node voting with $2i+1$ shares if they receive j shares of value v and t shares of value $-v$ such that $i+1 \geq j > t \geq 0$. This event occurs with probability $p(i) = \sum_{j=1}^{i+1} \sum_{t=0}^{j-1} \binom{D}{j}\binom{D-j}{t}/\binom{N}{j+t}$.

We have: $\binom{D}{j}\binom{D-j}{t}/\binom{N}{j+t} \leq \left(\frac{D}{N}\right)^{j+t}\binom{j+t}{j}$. Denote $a = D/N$, we will find the upper bound of $\sum_{j=1}^{i+1}\sum_{t=0}^{j-1} a^{t+j}\binom{j+t}{j}$.

Rewrite that expression as $\sum_{r=1}^{2i+1} a^r \sum_{i<j\wedge i+j=r}\binom{r}{i} \leq \sum_{r=1}^{2i+1} a^r \frac{1}{2}\sum_{i=0}^{r}\binom{r}{i} = \frac{1}{2}\sum_{r=1}^{2i+1}(2a)^r = \frac{a}{1-2a}\left(1-(2a)^{2i+1}\right)$.

Since a node sends $2i+1$ shares with probability γ_i, and we consider all possibilities of value i, this gives us the desired result $P_{un} = \sum_{i=0}^{k}\gamma_i p(i) \leq \frac{a}{1-2a}[1-\sum_{i=0}^{k}\gamma_i(2a)^{2i+1}]$. □

By Theorem 6.5, the quantity P_{un} increases when both values γ_i and D increase (and also i increases). We consider a graph with $N = 100$ and $k = 1$ (i.e., each node votes with one share or $2k+1 = 3$ shares). Fig. 6.3b shows the impact of the number of dishonest nodes D and the proportion γ_k of nodes voting with $2k+1$ shares on the probability of non-greedy vote disclosing. According to this result, the vote privacy decreases (i.e., value P_{un} increases) when both the proportion of nodes voting with $2k+1$ shares and the number of dishonest nodes increase.

Combining vote disclosing with and without certainty. The dishonest nodes may try to reveal a node vote either in certainty or uncertainty. Assume they use the vote disclosing rule with and without certainty. From the viewpoint of dishonest nodes, they always want their vote detection to be as certain as possible, i.e., they prefer a node vote being revealed with certainty than other cases. Hence their strategy is as follows: they first try to disclose a vote of node with certainty . If they do not succeed, for instance, because of lacking of messages, they will consider the way to detect that vote without certainty. It means that the probability for a vote disclosure in this case is $P_{com} = \max\{P_{ce}, P_{gr}, P_{un}\}$.

Accuracy. In this part, we evaluate the maximum and average impact on accuracy caused by the dishonest coalition when we deploy EPol on the graphs of $\mathcal{G}_2$. The dishonest nodes' capabilities are similar to the ones given in Definition 4.5 of Chapter 4. More precisely, a dishonest node

may affect the poll outcome with the following misbehaviors:

(1) It sends more than $k+1$ (but not greater than $2k+1$) identical shares.

(2) It inverts each receiving "+1"-share into a "−1"-share to decrease the collected data.

(3) In the broadcasting phase, it modifies the collected data of other honest node or sends a forged message.

(4) It broadcasts or forwards inconsistent data.

We show the impact of these misbehaviors in Theorems 6.6 and 6.7.

Theorem 6.6 (Maximum impact)**.** *Each dishonest node may affect the final result up to* $6k+4$.

Proof. In cases (1) and (2), the worst case of these misbehaviors occurs when the dishonest node sends $2k+1$ shares of value -1 and inverts all $2k+1$ receiving shares before summing. As a node is allowed to send only a set of shares summing to -1 or $+1$, its first and second acts respectively affect up to $|2k+1-(-1)| = 2k+2$ (if its vote is $+1$ but it generates $2k+1$ -1-shares) and $(2k+1)-(-2k-1) = 4k+2$ (if it gets $2k+1$ shares of "+1" and inverts all into "−1"-shares). In the misbehavior (3), a dishonest node only modifies the collected data of other honest node s such that the data is in the range $[-2k-1, 2k+1]$, otherwise its misbehavior will be detected. However this activity does not affect the final result since one node receives a direct message from s (if it is a neighbor of s) or receives m forwarding messages from neighbors in which at most $\lfloor (m-1)/2 \rfloor$ messages are corrupted, and majority messages ($\lceil (m+1)/2 \rceil$) contain correct data, thus an honest node always obtains the precise collected data of s. Therefore, the maximum impact from one dishonest node is $2k+2+4k+2 = 6k+4$. □

Note that from the result of Theorem 6.6, we get a value $\sigma = \frac{6k+4}{2N} = \frac{3k+2}{N}$. Function $\xi(k) = \frac{3k+2}{N} \leq 1/k$ for $k \leq \lfloor (-1+\sqrt{3N+1})/3 \rfloor$. It means $\xi(k)$ decreases as $1/k$. Thus, it is negligible. By Definition 4.2, the protocol is accurate.

Moreover, by Theorems 6.2 and 6.6, we see that the number of users voting for $2k+1$ shares affects to the privacy and accuracy of EPol. If that number increases, then the privacy increases but the accuracy decreases, and vice versa.

Theorem 6.7 (Average impact)**.** *Let* α *be the proportion of nodes voting for "+1". The average impact from a dishonest node is* $I_{avg} = \left[\sum_{i=0}^{k} \gamma_i (2i+1)\right] \cdot \left[1 + 2\sum_{i=0}^{k} \gamma_i \frac{i+\alpha}{2i+1}\right] + 1$.

Proof. Each node generates $i+1$ shares of its tendency, and i opposite shares; thus, the consumers receive a $+1$-share with probability $\sum_{i=0}^{k} \gamma_i [\alpha \frac{i+1}{2i+1} + (1-\alpha)\frac{i}{2i+1}] = \sum_{i=0}^{k} \gamma_i \frac{i+\alpha}{2i+1}$. In addition, as the average number of receiving shares is $\sum_{i=0}^{k} \gamma_i (2i+1)$, the average number of $+1$-shares is $\left[\sum_{i=0}^{k} \gamma_i (2i+1)\right] \cdot \left[\sum_{i=0}^{k} \gamma_i \frac{i+\alpha}{2i+1}\right]$. By altering any receiving $+1$-share into -1-share this impacts $2\left[\sum_{i=0}^{k} \gamma_i (2i+1)\right] \cdot \left[\sum_{i=0}^{k} \gamma_i \frac{i+\alpha}{2i+1}\right]$. Moreover, a dishonest node sends on average $\sum_{i=0}^{k} \gamma_i (2i+1)$ shares of -1 which affects by $1 + \sum_{i=0}^{k} \gamma_i (2i+1)$. Consequently, the total average impact from a dishonest node is $I_{avg} = \left[\sum_{i=0}^{k} \gamma_i (2i+1)\right] \cdot \left[1 + 2\sum_{i=0}^{k} \gamma_i \frac{i+\alpha}{2i+1}\right] + 1$. □

The quantity I_{avg} is minimized when all nodes generate the same number of shares, e.g., $2i+1$, and thus $I_{avg} = 2i+2+2(i+\alpha) = 2(2i+\alpha+1)$. In the worst case, a dishonest node always sends $2k+1$ shares, hence, the minimized average impact is $I_{avg} = 2(2k+\alpha+1)$.

Corollary 6.2. *If all nodes send* $2k+1$ *shares, then the biased final result is* $(2\alpha-1)N - (4k+2\alpha+2)D$.

Proof. The expected final outcome is $\alpha N.1+(1-\alpha)N.(-1) = (2\alpha-1)N$. By Theorem 6.7, total impact is $(4k+2\alpha+2)D$, this yields the proof. □

By Theorems 6.2–6.7 and Corollaries 6.1–6.2, for a fixed parameter k, the number of users voting with a high number of shares (e.g., $2k+1$ shares) affects the privacy and accuracy. More concretely, if we care about vote privacy, we should augment the number of nodes generating $2k+1$ shares since the probability of vote disclosing with certainty (P_{ce}) and with greedy uncertainty (P_{gr}) will decrease. But this rises up the probability P_{un} of non-greedy vote disclosing and the impact on the final outcome. In contrast, if we take care of the accuracy of the final result, we should decrease the number of nodes voting with $2k+1$ shares since that reduces the impact on the final outcome. It also decreases the probability of non-greedy vote detection. However this increases the probability of a node vote to be revealed with certainty and with greedy uncertainty.

Security. In this part, we compute the maximum number of dishonest nodes that EPol can tolerate.

Lemma 6.1. *A node decides correctly the collected data of an honest source s if it connects directly to s or there are at most $(m-1)/2$ dishonest neighbors preceding it (in the ordering w.r.t. source s).*

Proof. It is a trivial case when a node connects directly to source s. We only consider the case where a node $n \notin \Gamma(s)$. Since a node gets randomly m messages among $\beta_n(s)$ ones, and its decision is based on the majority appearance of a value, it is necessary less than $m/2$ messages are corrupted. This infers the desired result. □

Theorem 6.8 (Tolerance to Dishonest Nodes)**.** *The maximum number of dishonest nodes that EPol can tolerate is $\frac{m-1}{2}Diam(G)$.*

Proof. Recall the neighborhoods of a node, defined in Chapter 3: the set of direct neighbors of node n is $\Gamma_n^1 = \Gamma(n)$ and the collection of neighbors at distance $j > 1$ from n is $\Gamma_n^j = \{u \mid \delta(u,n) = j\} = \{u \mid u \in \Gamma_v \text{ where } v \in \Gamma_n^{j-1} \text{ and } u \notin \bigcup_{k<j} \Gamma_n^k\}$.

We define by $\varphi_n(s)$ the set of n's neighbors preceding n in the ordering w.r.t. source s. We have $|\varphi_n(s)| = \beta_n(s)$. Recall that $\delta(u,v)$ denote the distance, i.e., the length of the shortest path, between nodes u and v.

By Lemma 6.1, a node n decides correctly about a collected data of honest source s if $n \in \Gamma(s)$ or $|\varphi_n(s) \cap \mathcal{D}| \leq (m-1)/2$. To find out the highest number of dishonest nodes in the system we only consider a node $n \notin \Gamma(s)$. In general, a node $u \in \varphi_n(s)$ may be at any distance from s (even $\delta(u,s) \geq \delta(n,s)$). It means n may receive a corrupted data from a dishonest node at any distance from s. Since n receives at most $(m-1)/2$ corrupted messages, the necessary condition to guarantee n decides correctly s's collected data is there are at most $(m-1)/2$ dishonest friends of s at distance i, i.e., $|\Gamma_s^i \cap \mathcal{D}| \leq (m-1)/2$ where $1 \leq i \leq Rad(s,G)$ and $Rad(s,G) = \max_{u \in V}\{\delta(s,u)\}$.

This gives the maximum number of dishonest nodes, such that each node receives correct collected data of source s, is $D(s) = \sum_{i=1}^{Rad(s,G)} |\Gamma_s^i \cap \mathcal{D}| \leq \frac{m-1}{2}Rad(s,G)$. Considering all source nodes in the network, we have $D \leq \max_{s \in V}\{D(s)\} = \frac{m-1}{2}Diam(G)$.

An example of the graph which tolerates $D = \frac{m-1}{2}Diam(G)$ dishonest nodes is illustrated in Fig.6.4 for $m = 3$. □

Observation. It is also noted that our protocol can tolerate more than $\sqrt{N}$ dishonest nodes for a ring-based structure introduced in [82, 83]. Indeed, as this structure has the diameter $Diam(G) = \sqrt{N}$, and with parameter $m \geq 3$, the upper bound of D is not less than $\sqrt{N}$.

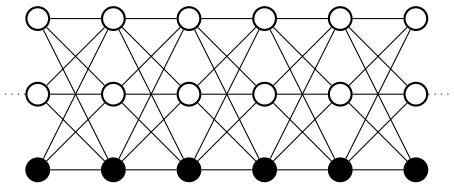


Figure 6.4: A graph where protocol tolerates $\frac{m-1}{2}Diam(G)$ dishonest nodes.

Corollary 6.3. *If $D \leq \frac{m-1}{2}Diam(G)$ then a node decides wrongly the collected data of some other node with the probability converging to 0 exponentially fast in N (and $Diam(G)$).*

Proof. We define by $\varphi_n(s)$ the set of n's neighbors preceding n in the ordering w.r.t. source s. By Lemma 6.1, if $|\varphi_n(s) \cap \mathcal{D}| \geq m/2$ then a node n may decide wrongly the collected data of source s (or could not make decision). We inspire the idea from [83] to compute the probability this event occurs by using standard Hoeffding bounds for sampling from a finite population without replacement (with the notice that $Diam(G) \leq N-1$):

$$p_s = Pr\Big[|\varphi_n(s) \cap \mathcal{D}| \geq m/2\Big] = Pr\Big[|\varphi_n(s) \cap \mathcal{D}| - \frac{D}{Diam(G)} \geq \frac{1}{Diam(G)}(\frac{m}{2}Diam(G) - D)\Big]$$
$$\leq \exp(\frac{-2}{Diam(G)}(\frac{m}{2}Diam(G) - D)^2) \leq \exp(\frac{-2}{N-1}(\frac{m}{2}Diam(G) - D)^2).$$

As $D < \frac{m}{2}Diam(G)$, the right-hand function tends to 0 exponentially fast in N (and $Diam(G)$). It implies the probability for a node decides wrongly the collected data of some source (or could not make decision) is

$$P_c = Pr\Big[\bigcup_{s \in V}\{|\varphi_n(s) \cap \mathcal{D}| \geq m/2\}\Big] \leq \sum_{s \in V} p_s \leq N \exp(\frac{-2}{N-1}(\frac{m}{2}Diam(G) - D)^2).$$

The right-hand function also tends to 0 exponentially fast in N. □

EPol vs. graphs. We examine the necessary and sufficient condition for our protocol to be deployed correctly in the following Theorem.

Theorem 6.9. *The properties of $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) are the necessary and sufficient conditions for EPol to be deployed correctly in the system without (resp. with) the presence of dishonest nodes.*

Proof. We here just show this theorem with graphs of $\mathcal{G}_2$. For graphs of $\mathcal{G}_1$, the proof is easily considered as its consequence. The sufficient condition ($\Leftarrow$) is proved by Theorems 6.1–6.8 since our protocol works correctly with graphs of $\mathcal{G}_2$ and preserves its properties such as correctness, privacy, accuracy, termination.

We only need to clarify the remaining proof of this theorem. Assume we have a general graph G. We give the proof by sketching step by step the requirements G should obtain so that all properties of protocol are guaranteed.

In the sharing phase, node n sends (or receives) at most $2k+1$ messages, i.e., $|\mathcal{S}_n| = |\mathcal{R}_n| \leq 2k+1$. In addition, it has to send an odd number of messages, thus, $|\mathcal{S}_n| = 2i+1$ where $0 \leq i \leq k$. As $|\mathcal{S}_n|$ and $|\mathcal{R}_n|$ might not be disjoint, then $d_n \geq 2k+1$. Therefore, to apply protocol correctly, G must have the property P_{g_1}. In addition, in the broadcasting phase of EPol, a node decides a data of some source s if it connects directly to s, or to at least m nodes which have preceding orders. This infers the graph must satisfy the m-broadcasting condition. Finally, the Lemma 6.1 shows the necessary property P_{g_3} that a graph must satisfy to deploy EPol. Accordingly, the graph G is a member of family $\mathcal{G}_2$ of graphs. □

6.4 Crash and message loss analysis

Nodes communicate by UDP which may suffer message loss on the communication channels. In addition, nodes may be unreliable, causing expected messages to be lost due to sender crashes. Assuming the presence of node crashes and message losses in the system, this part analyzes the effect of these factors on accuracy and termination of the protocol. Here we assume the system contains no dishonest nodes.

Impact on termination. In the following, we study the effect on termination of the protocol by computing the probability of a node failing to decide on the final result, i.e., it has not decided on at least one collected data of a certain source s.

We suppose that nodes crash with probability r (and never recover from a crash) and a message is lost (after transmitting from a sender) with probability l. We define a probability q for a node *fails to send shares* to its consumers in the sharing phase, i.e., $q = r + (1 - r)l$.

A node n *fails to forward* a collected data of source s to some node if it either: (i) crashed; (ii) has itself not decided that value; or (iii) has forwarded a message but it is lost while transmitting. In addition, a node n also *fails to decide* a collected data of source s if one of the following events occurs:

1. $n = s$: s fails to compute its collected data c_s.

2. $n \in \Gamma(s)$: n does not receive a broadcast message from s.

3. $n \notin \Gamma(s)$: n receives at most $(m-1)$ messages from neighbors (preceding it in the ordering w.r.t. source s), i.e., more than $\beta_n(s) - m$ (preceding) neighbors fail to forward the collected data.

Let e_{n_i} be the probability for a node n at distance i from source s to fail to forward a collected data of s. Let z_{n_i} be the probability for a node n at distance i from source s to fail to fail to decide a collected data of s.

We have $e_{n_i} = r + (1 - r)[z_{n_i} + (1 - z_{n_i})l]$

where:

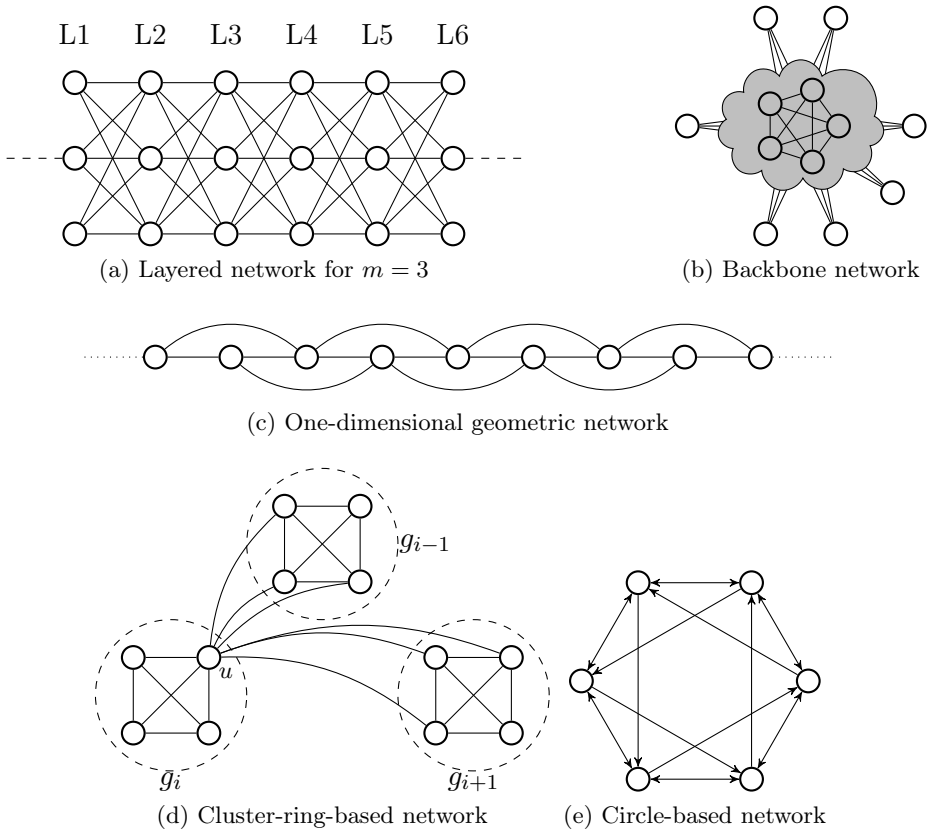
$z_{n_0} = z_s = \sum_{j=0}^{|\mathcal{R}_s|-1} \binom{|\mathcal{R}_s|}{j} (1-q)^j q^{|\mathcal{R}_s|-j}$,

$z_{n_1} = e_{n_0}$,

and $z_{n_i} = \sum_{j=0}^{m-1} \left[\binom{\beta_n(s)}{j} \prod_{t=1}^{j} (1 - e_{n_{t_{l_t}}}) \prod_{p=j+1}^{\beta_n(s)} e_{n_{p_{l_p}}} \right]$ where $i \geq 2$, $\{n_1, n_2, \ldots, n_{\beta_n(s)}\}$ and $\{l_1, l_2, \ldots, l_{\beta_n(s)}\}$ are respectively the sets of preceding friends of n (w.r.t. source s) and their corresponding distances to s. It is noted that l_j could be greater than i for $j = 1, 2, ...\beta_n(s)$.

In the worst case, the source s has $|\mathcal{R}_s| = 2k + 1$ and a node does not decide on a collected data of that node with a probability $z_{n_{Rad(s,G)}}$ where $Rad(s, G) = \max_{u \in V}\{\delta(s, u)\}$. Thus, a node does not decide on the final result if it has not decided on at least one collected data of a certain source, that is $z_{n_{Diam(G)}}$ where $Diam(G) = \max_{s \in V}\{Rad(s, G)\}$.

Impact on accuracy. A node crash affects the final result when that node has some unique information and they have not yet been replicated, typically the shares which are generated by node to represent its votes. It crashes (i) while sending its shares to consumers, or (ii) after summing up shares from producers. The former case affects to the final outcome up to $k + 1$ (if it crashes after sending k shares of $-v$). The later case affects up to $2k + 1$. Hence, the impact of such a crash on final result is bounded by $k + 1 + 2k + 1 = 3k + 2$.



(a) Layered network for $m = 3$

(b) Backbone network

(c) One-dimensional geometric network

(d) Cluster-ring-based network

(e) Circle-based network

Figure 6.5: Examples of networks satisfying the 3-broadcasting-source condition.

6.5 Particular graphs

The algorithm EPol requires the graphs satisfying the m-broadcasting property (see Definition 6.1). Here we illustrate some particular graphs of this family. The examples of these graphs for $m = 3$ are also depicted in Fig.6.5 and they can be generalized to any value m. Some of these graphs are given by the work [166].

(a) **Layered networks.** Each layer contains at least m nodes, and each node links to all other nodes in neighboring layer. The number of nodes in each layer could be different. For each source node, an (increased) ordering satisfying the m-broadcasting property w.r.t. this source is the ordering of the nodes with respect to the distance from the source.

(b) **Networks with a backbone.** The graph includes a *backbone* which is densely connected graph, and other nodes outside the backbone which connect directly to at least m nodes in the backbone. An ordering satisfying the m-broadcasting-source property w.r.t. one source is the ordering under the backbone subgraph, followed by all remaining nodes not in the backbone.

(c) **One-dimensional geometric networks.** All nodes are arranged along a line, and there is a connection between two nodes if their distance is smaller than a fixed threshold. Each node has at least m connections. An ordering of nodes w.r.t. one source is the order of their euclidean distance from the source.

(d) **Cluster-ring-based graph [24, 25, 65, 82, 83].** The N nodes are clustered into $r = \sqrt{N}$ *ordered* groups, from g_0 to g_{r-1}. Each group is a clique. A node n in group g_i also links to a fixed-size set $\mathcal{S}_n$ of nodes in the next group ($\mathcal{S}_n \subset g_{i+1 \bmod r}$), and a fixed-size set $\mathcal{R}_n$ of nodes in the previous group where $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k + 1$. Thus, all groups virtually form a ring with g_0 being the successor of g_{r-1}. In fact, this structure is a particular case of layered networks presented above in which each layer is a clique of size $\sqrt{N}$, and all layers virtually form a ring. Hence for each source node, an (increased) ordering satisfying the m-broadcasting property w.r.t. a source is the ordering of the nodes with respect to the distance from the source. It is also ordered based on the distance from the source and the direction of sending messages, e.g., from group g_i to $g_{i+1 \bmod r}$.

(e) **Circle-based networks.** Consider a graph $G_0 \in \mathcal{G}_2$ of size $N > 2k + 1$ where each node n has the set of consumers $\mathcal{S}_n$ which is the output of the function:

$$\begin{aligned} f\colon A &\to 2^A \\ n &\mapsto \{(n-1) \bmod N, (n+1) \bmod N, \ldots, (n+2k) \bmod N\} \end{aligned}$$

where $A = \{0, 1, 2 \ldots, (N-1)\}$. The set of producers is $\mathcal{R}_n = \{(n+1) \bmod N, (n-1) \bmod N, (n-2) \bmod N, \ldots, (n-2k) \bmod N\}$ of size $2k+1$. Figure 6.5e depicts an example of this graph with $N = 6$ and $k = 1$ in which the arrows express the direction of sending messages. For each source node, an (increased) ordering satisfying the m-broadcasting-source property w.r.t. this source is the ordering of the nodes with respect to the distance from the source.

All protocols in [24, 25, 65, 82, 83] cannot be deployed in this graph since the condition $\mathcal{S}_n \cap \mathcal{R}_n = \varnothing$ (proposed in these works) is not satisfied for every node n.

6.6 Summary and discussion

This chapter introduced the third distributed polling protocol, EPol, that assure vote privacy and limit the impact on accuracy of the polling outcome and particularly could be deployed on the family of social networks which satisfy the m-broadcasting property. In this protocol, each node can obtain a correct data of other node in the broadcasting procedure without requiring any verification process. In addition, we introduced some uncertain vote disclosing rules for dishonest node, and presented the probabilities of vote detection in these cases. We also analyzed the effect of message losses and crashes on accuracy and termination. Despite the use of richer social graph structures, the communication and spatial complexities of EPol are close to be linear. Next chapter, we will discuss the graph transformation problem that adds a minimum number of edges to a given graph while satisfying a threshold parameter.

Chapter 7

On Constrained Adding Friends in Social Networks

Contents

7.1 Problem statement . . . 100
7.1.1 Notations . . . 100
7.1.2 Problem definition . . . 100
7.2 Centralized protocol . . . 103
7.2.1 Protocol description . . . 103
7.2.2 Correctness . . . 106
7.2.3 Greedy algorithm . . . 109
7.3 Decentralized protocol . . . 112
7.3.1 Model . . . 113
7.3.2 Protocol description . . . 114
7.3.3 Correctness . . . 116
7.3.4 Algorithms comparison . . . 118
7.4 Experimental evaluation . . . 121
7.4.1 Datasets . . . 122
7.4.2 Experimental setup . . . 122
7.4.3 Results . . . 123
7.5 Conclusion . . . 125

The distributed polling protocols we presented in this thesis and other secret sharing based protocols (such as [24, 25, 65, 76, 82, 83]) often impose a threshold parameter on the number of friends, e.g., the minimum degree of the social graph should be not smaller than the given threshold. Often this condition is not satisfied by all social graphs. Yet we can reuse these graphs after some structural modifications consisting in adding new friendship relations. In this chapter, we provide the first definition and theoretical analysis of the "*adding friends*" problem. We formally describe this problem that, given a graph G and parameter c, asks for the graph satisfying the threshold c that results from G with the minimum of edge-addition operations. We present algorithms for solving this problem in centralized and decentralized networks. We show that no distributed algorithm is better than centralized one for all graph structures. In addition, we also prove that there is no best decentralized solution, i.e., any decentralized algorithm can be

worse than other decentralized one for some graphs, but it can be better in some other scenarios. An experimental evaluation on real-world social graphs demonstrates that our protocols are accurate and inside the theoretical bounds.

This chapter is structured as follows. Section 7.1 describes our adding friends problem. Section 7.2 presents our protocols for the centralized social network with their correctness properties. Section 7.3 introduces our distributed algorithms, and shows that no one amongst them is the best, as well as they cannot compete with the centralized solutions. Section 7.4 illustrates our experimental results. We conclude the chapter in Section 7.5.

7.1 Problem statement

7.1.1 Notations

We model a social network with an undirected connected graph $G = (V, E)$ as presented in Sub-section 3.2.1 of Chapter 3. Let c be an initial constant parameter in the graph G that is used to identify two kinds of nodes: weaker and normal nodes.

Definition 7.1. *Let $G = (V, E)$ be a graph. A node $n \in V$ is* weaker *(resp.* normal*) if $d_n < c$ (resp. $d_n \geq c$). A graph G is* c-degree *if all nodes are normal. Otherwise, it is called* non-c-degree *graph.*

In addition, we have the following monotonicity property of the c-degree graph.

Observation 7.1. *If a graph is c_1-degree, then it is also c_2-degree for all $c_2 \leq c_1$.*

Observation 7.2. *If a graph is non-c_1-degree, then it is also non-c_2-degree for all $c_2 \geq c_1$.*

A set of all weaker nodes is denoted by $\mathcal{W} = \{n \in V \mid d_n < c\}$. A weaker node $n \in \mathcal{W}$ has to add one or more edges to become a normal one. Thus, we define y_n as the number of these edges, i.e., $y_n = c - d_n$. A weaker node n may create links with some other weaker ones if there are no connection between n and them. We call them *available weaker nodes* (i.e., *potential friends*) of n, and denote the set of these nodes by $\mathcal{D}_n = \{v \in \mathcal{W} \mid e(n, v) = 0\}$ and write its size by x_n.

7.1.2 Problem definition

Given a parameter c and a non-c-degree graph $G = (V, E)$ where $c < N$. Our purpose is to transform G into a *c-degree* graph $G' = (V, E')$ that is structurally similar to G by using a graph-modification operation on G. The output G' must have the same set of nodes as G.

It should be noted that we only use edge-addition operation to modify G into G'. Moreover, once one friendship link has been established, it cannot be removed or changed later. We assume that these new friendship links will be accepted by all user nodes as they are aware these new links will enable them to achieve some common functions (e.g., polling process).

Before presenting formally the definition of our adding friends problem, we describe the difference between two graph structures as follows.

Definition 7.2. *Given two graphs $G = (V, E)$ and $G' = (V, E')$. The* structural difference Φ *between G and G' is the difference between sets of edges. More formally,*

$$\Phi(G', G) = |E' \setminus E| + |E \setminus E'| \tag{7.1}$$

As we only use edge-addition operation to modify graph, $E' \supseteq E$ and G' is a *supergraph* of G. Consequently

$$\Phi(G', G) = |E' \setminus E| + |E \setminus E'| = |E'| - |E| \tag{7.2}$$

Naively, to construct G' from G, one weaker node could create friendship with anyone by accumulating more and more friends until it becomes normal. However, this naive approach may be less accurate and unfair:

(i) An unnecessary large number of friendship links may be added, namely it is a 2-approximation algorithm. For instance, assume we have w unconnected weaker nodes and each weaker node needs one new link to become normal node. In this case, the naive approach may add up to w new links, whereas $\lceil \frac{w}{2} \rceil$ is sufficient if we link weaker nodes together;

(ii) The process may not be fair in the sense that some user nodes could get more new links than others. We also observe that the naive approach takes time $\mathcal{O}(c|\mathcal{W}|)$ (it takes $\mathcal{O}(N^2)$ in the worst case when $|\mathcal{W}| = N$ and $c = N - 1$).

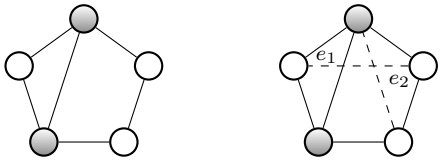
Problem (AddFriends). *Given a graph $G = (V, E)$ and a constant parameter c, construct a c-degree graph $G' = (V, E')$ by using only an edge-addition operation such that $\Phi(G', G)$ is minimized.*

Minimizing $\Phi(G', G)$ is equivalent to minimize the degree difference between G' and G. Indeed, as each edge is incident to exactly two vertices, we have $\sum_n d_n = 2|E|$ and $\sum_n d'_n = 2|E'|$, where d_n and d'_n are the degree of node n in G and G' respectively. Therefore, Formula (7.2) is rewritten as $\Phi(G', G) = \frac{1}{2} \cdot \sum_n (d'_n - d_n)$.

On the first view, our AddFriends problem seems generally related to the b-Matching problem [64, 119, 176, 178], or the solution could be the reduction of some b-matching algorithms proposed in [14, 139] but it turns out AddFriends problem and b-Matching problem are truly different. Indeed, the concept of a b-matching was defined by Edmonds [64] as follows: For each node $n \in V$ let $\delta(n)$ denote the set of edges in E which link n. For each vector $x = (x_e : e \in E)$ and a subset E' of E, let $x(E') = \sum\{x_e : e \in E'\}$. Let $b = (b_n : n \in V)$ be a positive integer vector. A *b-matching* of G is a nonnegative integer vector $x = (x_e : e \in E)$ such that $x(\delta(n)) \leq b_n \ \forall n \in V$. From this definition, in the b-Matching problem, we have to use *edge-deletion* operation to modify G into a *subgraph* G' in such a way that each node n in the output must satisfy the condition $d_n \leq b_n$. Conversely, in the AddFriends problem we only allow *edge-addition* operation to modify G into a *supergraph* G' and each node n has to satisfy reverse condition for degree value $d_n \geq b_n$ (and $b_n = c \ \forall n \in V$). Moreover, the b-matching problem on the complement graph is also not in our consideration as the generation of the complement graph is based on the removing and adding edges from the initial graph. These differences are also applied for the generalization of the b-Matching problem proposed by Lovász [125, 126], General Factor problem, where each vertex's degree of the output graph must belong to a list of possible predefined values called *degree list*. It is also known that by results given in [51, 133], General Factor problem (where all degree lists may not contain gaps of length greater than one) and graph editing problem (where all degree lists are singleton) can be solved in polynomial time. Nevertheless, to the best of our knowledge, there is no efficient algorithm solving this problem with precise complexity.

To solve AddFriends problem, we only examine the following two types of edge related to: (i) two weaker nodes, (ii) one weaker node and one normal node. It is noted that added edges

is a *dynamic process*, and during this process, a weaker node may become a normal node. We denote by κ and λ the number of added edges of type (i) and (ii) respectively with respect to the dynamic process (not with respect to the original static graph). In other words, an added edge only counts as a "κ" edge if the two vertices are weak at the time the edge is added, and it counts as a "λ" edge if one vertex is weak and other other is normal at the time the edge is added. For instance, given $c = 3$, Fig. 7.1 illustrates an initial graph G and an output graph G', where weaker and normal nodes are drawn in white and gray color respectively, and dashed lines are added edges. In this case $\kappa = 1$ and $\lambda = 1$.



(a) Graph G (b) Output G'

Figure 7.1: Example of value κ and λ.

Lemma 7.1. *An algorithm solves the adding friends problem, i.e., obtaining a minimum value of Φ, if and only if it has a maximum value of κ, or minimum value of λ (w.r.t. all possible values κ or λ obtained by any other algorithms).*

Proof. The total degree of all weaker nodes should be increased is $\sum_{n\in\mathcal{W}} y_n$. Furthermore, as one edge is incident to exactly two nodes, to add κ (resp. λ) edges established by pairs of two weaker nodes (resp. pairs of one weaker node and one normal node), total degree of weaker nodes should be extended more 2κ (resp. λ). It follows $\sum_{n\in\mathcal{W}} y_n = 2\kappa + \lambda$. Thus, $\Phi = \kappa + \lambda = \kappa + \sum_{n\in\mathcal{W}} y_n - 2\kappa = \sum_{n\in\mathcal{W}} y_n - \kappa$, and $\Phi = \kappa + \lambda = \frac{1}{2}\left(\sum_{n\in\mathcal{W}} y_n - \lambda\right) + \lambda = \frac{1}{2}\left(\sum_{n\in\mathcal{W}} y_n + \lambda\right)$. Because $\sum_{n\in\mathcal{W}} y_n = \sum_{n\in\mathcal{W}}(c - d_n) = c|\mathcal{W}| - \sum_{n\in\mathcal{W}} d_n$ is a constant, two equations above yield the claim. □

Lemma 7.1 means: to achieve the minimum number of added edges, we have to target $\mathcal{W}$ by trying to create connections amongst nodes inside $\mathcal{W}$ (i.e., increasing κ and decreasing λ).

In this chapter, we study the AddFriends problem in two models:

- **Centralized model**: There exists a special user node (e.g., super node or central authority) who has the full knowledge and control over the social graph. Given a parameter c, this super node can detect weaker nodes and accordingly add for them necessary friendship links. This model encompasses existing social platforms such as Facebook, LinkedIn, etc. To our best knowledge, there is no known strategy to compute the minimized value Φ.

- **Decentralized model**: Each user node has only partial knowledge over the social graph. Solving the AddFriends problem in this model is more challenging because it is generally not possible for a node to know the whole network and gather information of other nodes to request/accept friendship relations. Therefore, users can only rely on their local knowledge to choose prospective friends.

7.2 Centralized protocol

In this section, we first present our algorithm, AlgoCen, based on centralized model. Then we show the correctness of this algorithm and provide a theoretical analysis of its complexity. Finally we introduce two intuitive greedy algorithms with their complexities, and describe the comparison in the aspect of accuracy and complexity between greedy algorithms and AlgoCen.

7.2.1 Protocol description

The adding friends protocol, *AlgoCen*, is composed of two cases (see Algorithm 4). If $c = N - 1$ we apply the naive approach (lines 1–3), i.e., each weaker node simply links to any other nodes. Otherwise, we consider the following stages.

Stage 1 (lines 4–27 in Algorithm 4). The system gets some information related to each weaker node n such as the set of potential friends $\mathcal{D}_n \subseteq \mathcal{W}$ and its size x_n, number of new friends to be added more $y_n = c - d_n$. The boolean variable $flag$ is used to avoid computing $\mathcal{D}_n$ twice at the initial. The idea behind this algorithm is that we assign higher selection priority to weaker nodes having less number of potential friends than they require. In other words, first a set $\mathcal{R} = \{n \in \mathcal{W} \mid y_n \geq x_n \text{ and } x_n \geq 0\}$ is examined, and later we investigate the set $\mathcal{U} = \mathcal{W} \setminus \mathcal{R}$. After generating $\mathcal{R}$ (line 8), we evaluate the following two cases:

1. $\mathcal{R} \neq \varnothing$ (lines 10–21): each node $n \in \mathcal{R}$ where $x_n > 0$ links to all nodes v in $\mathcal{D}_n$. The procedure `ConnectUpdateInfo`(n, v) (line 15) simply creates a connection between n and v, then updates their information, e.g., d_n, $\mathcal{D}_n$, x_n, y_n, d_v, $\mathcal{D}_v$, x_v, y_v. (For the sake of simplicity, we do not describe this procedure in the algorithm.)

   After the link between n and v is generated, if v's degree satisfies the threshold condition, i.e., $d_v = c$, nodes v is removed out of $\mathcal{W}$ and also out of $\mathcal{R}$ if it is currently inside $\mathcal{R}$ (lines 16–18).

   In addition, for a node $n \in \mathcal{R}$ where $x_n = 0$, it may not connect to any other weaker nodes as they already linked to n or became normal ones in the dynamic process. Therefore, n has to request for friendship relations to normal nodes. We use a set $\mathcal{T}$ to hold all of these weaker nodes which have no potential friends. This set will be resolved in next stage. Thus, in this case node n is put into $\mathcal{T}$ (line 19).

   It is noted that all nodes in $\mathcal{R}$ are finally removed out of $\mathcal{R}$ and $\mathcal{W}$ (lines 20–21).

2. $\mathcal{R} = \varnothing$: We first introduce one criterion to evaluate the selection priority of nodes. For each (weaker or normal) node n, we define a value t_n called *score* as follows:

$$t_n = \begin{cases} \dfrac{y_n}{x_n} & \text{if } x_n \neq 0 \text{ and } n \text{ is weaker} \\ 0 & \text{if } x_n = 0 \text{ and } n \text{ is weaker} \\ -1 & \text{otherwise} \end{cases} \tag{7.3}$$

   This value expresses the level of aspiration of n to make connection to other nodes when it currently needs adding more y_n links but has a limited number x_n of weaker options. The higher score value, the higher selection priority. (If two nodes have the same score value then the one with smaller identity is considered to be higher priority.) We say a tuple of nodes (u_1, v_1) is *better* than a tuple (u_2, v_2) w.r.t. score value (assume initially $t_{u_1} \geq t_{v_1}$ and $t_{u_2} \geq t_{v_2}$) iff: (i) $t_{u_1} > t_{u_2}$, or, (ii) $t_{u_1} = t_{u_2}$ and $t_{v_1} \geq t_{v_2}$. A *best* tuple of nodes w.r.t. score value is the one that is better than any other tuples of nodes.

Algorithm 4: CENTRALIZED ADDING FRIENDS ALGORITHM - ALGOCEN

Input: a non-c-graph $G = (V, E)$ with a parameter c
Output: a c-degree graph $G' = (V, E')$.

```
1  if c = N − 1 then
2      Apply the naive approach for G
3      exit
4  Get information about D_n, x_n, y_n for all n ∈ W
5  flag ← true
6  T ← ∅
7  while W ≠ ∅ do
8      R ← {n ∈ W | y_n ≥ x_n and x_n ≥ 0}
9      if R ≠ ∅ then
10         foreach n ∈ R do
11             if x_n > 0 then
12                 if flag = false then
                       D_n ← {u ∈ W | e(n, u) = 0}
13                 else flag ← false
14                 foreach v ∈ D_n do
15                     ConnectUpdateInfo(n, v) /* create connection, update d_n, x_n, y_n, D_n, d_v, x_v, y_v, D_v */
16                     if d_v = c then
17                         if v ∈ R then R ← R \ {v}
18                         W ← W \ {v}
19             if d_n < c then T ← T ∪ {n}
20             R ← R \ {n}
21             W ← W \ {n}
22     else
23         Compute score value for all weaker nodes
24         Find best tuple of weaker nodes (n, v) w.r.t. score value and e(n, v) = 0
25         ConnectUpdateInfo(n, v)        /* create connection, update d_n, x_n, y_n, D_n, d_v, x_v, y_v, D_v */
26         if (d_n = c) then W ← W \ {n}
27         if (d_v = c) then W ← W \ {v}
28 foreach n ∈ T do
29     Create (c − d_n) links between n and arbitrary (c − d_n) nodes v ∈ V s.t. e(n, v) = 0
```

In Algorithm 4, for this case $\mathcal{R} = \varnothing$ (lines 22–27): we first compute score values for all weaker nodes, then discover a best tuple of weaker nodes (n, v) w.r.t. score value (and they currently do not link to each other). Eventually we create a connection between them.

We repeat above activities until $\mathcal{W}$ is empty.

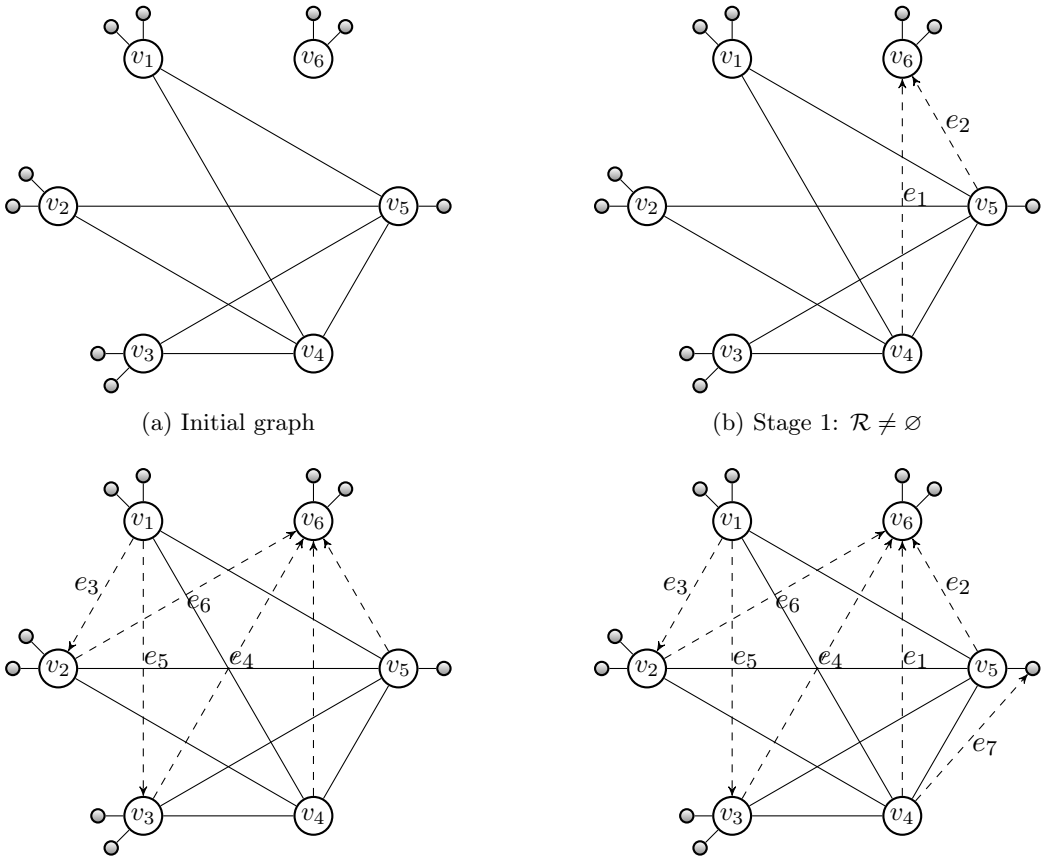
Stage 2 (lines 28–29 in Algorithm 4). This stage simply connects each node $n \in \mathcal{T}$ (if $\mathcal{T} \neq \varnothing$) generated from last stage to $(c - d_n)$ normal ones which currently do not have any connection to it.

In order to illustrate how Algorithm 4 works, we examine Example 7.1 below.

Example 7.1. Given $c = 6$ and the initial graph depicted in Fig. 7.2a.[15] All initial information of weaker nodes are as follows:
for v_1: $\mathcal{D}_1 = \{v_2, v_3, v_6\}$, $x_1 = 3$, $y_1 = 2$; for v_2: $\mathcal{D}_2 = \{v_1, v_3, v_6\}$, $x_2 = 3$, $y_2 = 2$;
for v_3: $\mathcal{D}_3 = \{v_1, v_2, v_6\}$, $x_3 = 3$, $y_3 = 2$; for v_4: $\mathcal{D}_4 = \{v_6\}$, $x_4 = 1$, $y_4 = 2$;
for v_5: $\mathcal{D}_5 = \{v_6\}$, $x_5 = 1$, $y_5 = 2$; for v_6: $\mathcal{D}_6 = \{v_1, v_2, v_3, v_4, v_5\}$, $x_6 = 5$, $y_6 = 4$.

In stage 1, we have $\mathcal{R} = \{v_4, v_5\}$. We take out randomly node by node from $\mathcal{R}$ and connect it with all available weaker nodes. So v_4 links to v_6 and v_5 also links to v_6 by edges e_1 and e_2 (see

[15] For the sake of simplicity, throughout this work, we depict only subgraphs of the original connected graph.



(a) Initial graph

(b) Stage 1: $\mathcal{R} \neq \varnothing$

(c) Stage 1 (cont.): $\mathcal{R} = \varnothing$

(d) Stage 2

Figure 7.2: AlgoCen algorithm example.

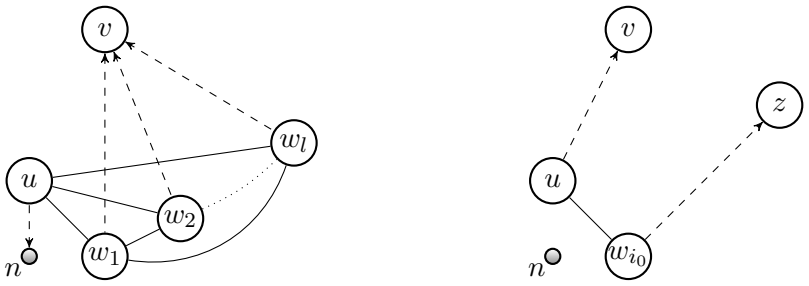
Fig. 7.2b) then they are eliminated from $\mathcal{R}$. The updated information of weaker nodes (where $\mathcal{W} = \{v_1, v_2, v_3, v_4, v_6\}$) are:
for v_1: $\mathcal{D}_1 = \{v_2, v_3, v_6\}$, $x_1 = 3$, $y_1 = 2$; for v_2: $\mathcal{D}_2 = \{v_1, v_3, v_6\}$, $x_2 = 3$, $y_2 = 2$;
for v_4: $\mathcal{D}_4 = \varnothing$, $x_4 = 0$, $y_4 = 1$; for v_3: $\mathcal{D}_3 = \{v_1, v_2, v_6\}$, $x_3 = 3$, $y_3 = 2$;
for v_6: $\mathcal{D}_6 = \{v_1, v_2, v_3\}$, $x_6 = 3$, $y_6 = 2$.

Since $x_4 = 0$ and $y_4 = 1$, node v_4 is moved from $\mathcal{R}$ to other set $\mathcal{T}$ (and considered later). Moreover, $\mathcal{R} = \varnothing$ as $y_i < x_i$, for all $i \in \{1, 2, 3, 6\}$. We compute score values for each weaker node to choose the best tuple of nodes (w.r.t. score value). Since $t_i = \frac{y_i}{x_i} = \frac{3}{2}$ for all i, we can opt any tuple, e.g., (v_1, v_2), to link up together by edge e_3 (see Fig. 7.2c).

Similarly, we update the information of weaker nodes again as follows:
for v_1: $\mathcal{D}_1 = \{v_3, v_6\}$, $x_1 = 2$, $y_1 = 1$; for v_2: $\mathcal{D}_2 = \{v_3, v_6\}$, $x_2 = 2$, $y_2 = 1$;
for v_3: $\mathcal{D}_3 = \{v_1, v_2, v_6\}$, $x_3 = 3$, $y_3 = 2$; for v_6: $\mathcal{D}_6 = \{v_1, v_2, v_3\}$, $x_6 = 3$, $y_6 = 2$.

We obtain $\mathcal{R} = \varnothing$ again and select tuple (v_3, v_6) as the best tuple w.r.t. score value, before generating an edge e_4. Repeating this process until $\mathcal{W} = \varnothing$, we get edges e_5,e_6 as presented in Fig. 7.2c.

In Stage 2, as $\mathcal{T} = \{v_4\}$, v_4 links to any other normal node (which is currently not connected to v_4), e.g., v_5, by friendship relation e_7. After this step, we have the final output graph illustrated in Fig. 7.2d. □



(a) In algorithm AlgoCen (b) In algorithm A'

Figure 7.3: Node v links to u in A' and but not do in AlgoCen.

7.2.2 Correctness

This section first analyzes the accuracy of AlgoCen and then presents its time complexity.

Theorem 7.1. AlgoCen *produces the optimal solution.*

Proof. Assume the contrary, i.e., there exists an algorithm A' different from AlgoCen such that when these algorithms are deployed into a certain scenario, we obtain $\Phi' < \Phi$, where Φ and Φ' are respectively the number of added edges produced by AlgoCen and A'.

Let κ and λ (resp. κ' and λ') respectively be the number of added edges between two weaker nodes, and the number of added edges between one weaker node and one normal node produced by algorithm AlgoCen (resp. A').

From the proof of Lemma 7.1: $\Phi = \sum_{n\in\mathcal{W}} y_n - \kappa = \frac{1}{2}(\sum_{n\in\mathcal{W}} y_n + \lambda)$ and $\Phi' = \sum_{n\in\mathcal{W}} y_n - \kappa' = \frac{1}{2}(\sum_{n\in\mathcal{W}} y_n + \lambda')$. Because $\sum_{n\in\mathcal{W}} y_n$ is constant, $\Phi' < \Phi$ iff $\kappa' > \kappa$ (or $\lambda' < \lambda$). W.l.o.g., suppose $\Phi' = \Phi - 1$, $\kappa' = \kappa + 1$, $\lambda' = \lambda - 2$. Since $\kappa' > \kappa$, there exists a weaker node u such that it has greater number of ading edges inside $\mathcal{W}$ in algorithm A' than the one in AlgoCen, i.e., there exists $v \in \mathcal{W}$ such that u connects to v in algorithm A' but not in algorithm AlgoCen (see Fig. 7.3).

Assume after deploying AlgoCen, v links to a set of weaker nodes $P = \{w_1, w_2, ..., w_l\}$, where $w_i \neq u, \forall i = 1, 2, ..., l$, and u connects to a certain normal node n as depicted in Fig. 7.3a.

In the process of Stage 1 of AlgoCen, because u wants to connect to some weaker nodes but there are not enough weaker nodes for it to create connection, hence, $x_u < y_u$, i.e., $t_u > 1$.

For node v, since $y_v = |P| = l$, and by hypothesis that in algorithm A' it can connect to u, i.e., $x_v \geq l + 1 > y_v$, thus $t_v < 1$.

It implies, by AlgoCen, (i) $t_{w_i} \geq 1$, v is chosen by w_i; (ii) at the time w_i processes to link to v, w_i already connects to u (for all $i = 1, 2, ..., l$) and w_i also connects to w_j (for all $i \neq j$).

Indeed, in AlgoCen, node with score value $t \geq 1$ has higher priority to be taken and connected to other nodes than nodes with score $t < 1$. In the dynamic process, before w_i links to v, we have: v has $t_v < 1$, u has $t_u \geq 1$.

Therefore, in case (i): if $t_{w_i} < 1$ then in AlgoCen, u should be chosen to process before node w_i does (since it has the highest priority at that time), and thus, it can connect to the weaker node v instead of joining to the normal one n. Contradiction!; in case (ii): at the time before w_i links to v, if w_i did not have a link with u then w_i and u can connect to each other, and w_i does not consider to connect to v. Contradiction!.

It is easy to see at that time w_i already linked to w_j since otherwise w_i can request to w_j to link together, and does not do with v. Fig. 7.3a depicts the adding friend process where w_i already linked to u and to other node w_j $(i \neq j)$.

In any algorithm, because total added degree from weaker nodes is a constant ($\sum_{n\in\mathcal{W}} c - d_n$), and in A', v is a friend of u, hence, there exists a node $w_{i_0} \in P$ (showed in Fig. 7.3b) such that by A' it is not connected to v (so that the total added degree in V is still l).

It implies w_{i_0} must connect to other weaker node z to obtain the same number of added edges (between w_{i_0} and other weaker nodes) as the one accomplished by AlgoCen.

It follows that in AlgoCen, before linking to v (and already linked to u), node w_{i_0} has the number of available weaker nodes $x_{w_{i_0}}$ greater than the number of weaker nodes $y_{w_{i_0}}$ to be added more, e.g., it chooses one node between v and z to link. It infers that $t_{w_{i_0}} < 1$ at that time point. Contradiction! □

Corollary 7.1. AlgoCen *produces value* $\Phi = \frac{1}{2}\Big[\sum_{n\in\mathcal{W}} y_n + \sum_{j=1}^{p}\sum_{n_i\in\mathcal{R}_j}(y_{n_i} - x_{n_i})\Big]$.

Proof. In Stage 1 of AlgoCen, we see that there is a repeated process of dividing a set $\mathcal{W}$ of weaker nodes into some subsets.

Indeed, firstly, set $\mathcal{W}$ is divided into two subsets $\mathcal{R}$ and $\mathcal{U}$ where $\mathcal{R}$ contains only weaker nodes satisfied condition $y_n \geq x_n \geq 0$, and $\mathcal{U}$ comprises the remaining weaker nodes m where $y_m < x_m$.

If $\mathcal{R} \neq \varnothing$, each node $n \in \mathcal{R}$ will add edges to x_n available weaker nodes in $\mathcal{W}$ (including d_{n_1} edges between it and nodes in $\mathcal{R}$ and d_{n_2} edges between it and ones in $\mathcal{U}$ where $d_{n_1} + d_{n_2} = x_n$). It then is put into the set $\mathcal{T}$ before making friendship relations with $y_n - x_n$ normal nodes later.

If $\mathcal{R} \neq \varnothing$, we create connection between two nodes having the highest score rate. In any case of $\mathcal{R}$, set $\mathcal{W}$ leads to another subset, e.g., $\mathcal{W}_2$, where the process is repeated. That set is divided into two subsets of nodes: a set $\mathcal{R}_2$ contains all node n having $y_n \geq x_n$, and a set $\mathcal{U}_2$ includes nodes m where $y_m < x_m$. Then $\mathcal{W}_2$ becomes a subset $\mathcal{W}_3$.

In general, a set $\mathcal{W}_i$ will generate two subsets $\mathcal{R}_i$ and $\mathcal{U}_i$. If $\mathcal{W}_i \neq \varnothing$, each node $n \in \mathcal{R}_i$ links to d_{n_1} nodes in $\mathcal{R}_i$ and d_{n_2} nodes in $\mathcal{U}_i$ where $d_{n_1} + d_{n_2} = x_n$, eventually it is put in $\mathcal{T}$. Otherwise, we make a link for best pair of weaker nodes w.r.t. score value which currently do not connect to each other. After all, $\mathcal{W}_i$ induces $\mathcal{W}_{i+1}$. Here we denote $\mathcal{W}_1 = \mathcal{W}$, $\mathcal{R}_1 = \mathcal{R}$, and $\mathcal{U}_1 = \mathcal{U}$.

That operation is repeated in a limited number of times (as the number of nodes is finite), until the set $\mathcal{W}$ could not be separated anymore.

Assume after p^{th} division, the process terminates, i.e., $\mathcal{W}_{p+1} = \varnothing$.

In Stage 2, all nodes in $\mathcal{T}$ has to connect to normal ones. It could be easy to see that:

$$|\mathcal{T}| = \sum_{n_1\in\mathcal{R}_1}(y_{n_1} - x_{n_1}) + \sum_{n_2\in\mathcal{R}_2}(y_{n_2} - x_{n_2}) + \cdots + \sum_{n_p\in\mathcal{R}_p}(y_{n_p} - x_{n_p}) = \sum_{j=1}^{p}\sum_{n_i\in\mathcal{R}_j}(y_{n_i} - x_{n_i})$$

By Lemma 7.1 with $|\mathcal{T}| = \lambda$, we have:

$$\Phi = \kappa + \lambda = \frac{1}{2}\Big(\sum_{n\in\mathcal{W}} y_n + \lambda\Big) = \frac{1}{2}\Big[\sum_{n\in\mathcal{W}} y_n + \sum_{j=1}^{p}\sum_{n_i\in\mathcal{R}_j}(y_{n_i} - x_{n_i})\Big]$$

□

Proposition 7.1 (Time Complexity)**.** *The time complexity of* AlgoCen *in the worst* (*resp. best*) *case is* $\mathcal{O}(N^4)$ (*resp.* $\mathcal{O}(cN)$).

Proof. If $c = N - 1$, the naive approach takes time $\mathcal{O}(c|W|)$, and that is $\mathcal{O}(cN)$ if $|W| = N$. If $c \neq N - 1$ then: in Stage 1, a set of weaker node $\mathcal{W}_1 = \mathcal{W}$ is divided into $\mathcal{R}_1$ and $\mathcal{U}_1$ (see Corollary 7.1 for more information of the division and denotation $\mathcal{W}_i$, $\mathcal{R}_i$, $\mathcal{U}_i$). We consider two cases as follows:

(i) $\mathcal{R}_1 \neq \varnothing$: each node $n_1 \in \mathcal{R}_1$ is moved into set $\mathcal{T}$ if $x_{n_1} > 0$, and contrary, it will be linked to other nodes in $\mathcal{W}_1$. The best situation is that all nodes in $\mathcal{R}_1$ are moved to $\mathcal{T}$, and it takes totally $\mathcal{O}(\sum_{n_1 \in \mathcal{R}_1} y_{n_1}) = \mathcal{O}(\sum_{n_1 \in \mathcal{R}_1}(c - d_{n_1})) = \mathcal{O}(c|\mathcal{R}_1|)$. The worst situation in this subcase is that all nodes in $\mathcal{R}_1$ are linked to other nodes in $\mathcal{D}_{n_1}$, and $\mathcal{D}_{n_1} = \mathcal{W}_1$, thus the time complexity is $\mathcal{O}((\sum_{n_1 \in \mathcal{R}_1} y_{n_1})|\mathcal{W}_1|) = \mathcal{O}((\sum_{n_1 \in \mathcal{R}_1}(c - d_{n_1}))|\mathcal{W}_1|) = \mathcal{O}(c|\mathcal{R}_1||\mathcal{W}_1|)$.

(ii) $\mathcal{R}_1 = \varnothing$: To get two highest-score weaker nodes, we order the list of weaker nodes with respect to score value (which takes $\mathcal{O}(|\mathcal{W}_1| \log |\mathcal{W}_1|)$), and try to find best tuple (u, v) w.r.t. score value and they are not connected to each other. In the best situation where all weaker nodes have different scores and each weaker node does not have connections to all other weaker ones, it takes $\mathcal{O}(|\mathcal{W}_1|)$ in time. However, at the worst case scenario where all nodes have the same score value and we have to compare all possible pairs of weaker nodes and find the best pair, the complexity will be $\mathcal{O}(|\mathcal{W}_1|^2)$. Therefore, for this value of $\mathcal{R}_1$, we obtain the complexity for best and worst case respectively are $\mathcal{O}(|\mathcal{W}_1| \log |\mathcal{W}_1| + |\mathcal{W}_1|) = \mathcal{O}(|\mathcal{W}_1| \log |\mathcal{W}_1|)$, and $\mathcal{O}(|\mathcal{W}_1|^2 + |\mathcal{W}_1| \log |\mathcal{W}_1|) = \mathcal{O}(|\mathcal{W}_1|^2)$.

For any case of $\mathcal{R}_1$, we then get a set $\mathcal{W}_2 \subseteq \mathcal{W}_1$. We find out a subset $\mathcal{R}_2 \subseteq \mathcal{W}_2$: if $\mathcal{R}_2 \neq \varnothing$, we make connections for nodes in $\mathcal{R}_2$; or conversely, we discover the best pair of two weaker nodes w.r.t. score value. Similar to result for first division, we have the complexity in the best case and worst case respectively: (i) if $\mathcal{R}_2 \neq \varnothing$: $\mathcal{O}(c|\mathcal{R}_2|)$ and $\mathcal{O}(c|\mathcal{R}_2||\mathcal{W}_2|)$; (ii) if $\mathcal{R}_2 = \varnothing$: $\mathcal{O}(|\mathcal{W}_2| \log |\mathcal{W}_2|)$ and $\mathcal{O}(|\mathcal{W}_2|^2)$.

This dividing process is repeated until $\mathcal{W}$ is empty. In stage 2, we need to enumerate all items in $\mathcal{T}$ which size is at most $|\mathcal{W}|$, and thus, time complexity for this stage is $\mathcal{O}(|\mathcal{W}|)$.

For the entire process, we see that:

a) The best case of the algorithm: in each i^{th} iteration of stage 1, we get $\mathcal{R}_i \neq \varnothing$ and all nodes in $\mathcal{R}_i$ are moved into set $\mathcal{T}$, i.e., $\bigcup \mathcal{R}_i = \mathcal{W}$. Hence, the overall time complexity of the algorithm for this situation is $\mathcal{O}(c \sum_i |\mathcal{R}_i| + |\mathcal{W}|) = \mathcal{O}(c|\mathcal{W}| + |\mathcal{W}|) = \mathcal{O}(c|\mathcal{W}|)$.

b) The worst case of the algorithm: in each i^{th} iteration of stage 1, we get $\mathcal{R}_i = \varnothing$ and we have to discover all possible pairs of weaker nodes to find the best one which does not have connection together (with complexity $\mathcal{O}(|\mathcal{W}_i|^2)$). We see that, each weaker node n has to create $(c - d_n)$ links to other $(|\mathcal{W}| - 1)$ ones, and one link is incident to precisely two weaker nodes, the maximum number of links (and also number of iterations in stage 1) is $\frac{1}{2}\sum_{n \in \mathcal{W}}(c - d_n) = \mathcal{O}(c|\mathcal{W}|)$. Moreover, for a set of size $|\mathcal{W}|$, the maximum number of pairs we could have is $\binom{|\mathcal{W}|}{2} = \mathcal{O}(|\mathcal{W}|^2)$. Thus, the number of iteration is $\max\{\mathcal{O}(c|\mathcal{W}|), \mathcal{O}(|\mathcal{W}|^2)\}$. The complete complexity in this case is $\max\{\mathcal{O}(c|\mathcal{W}|^3), \mathcal{O}(|\mathcal{W}|^4)\} + \mathcal{O}(|\mathcal{W}|) = \max\{\mathcal{O}(c|\mathcal{W}|^3), \mathcal{O}(|\mathcal{W}|^4)\}$.

If all nodes are weaker, i.e., $|\mathcal{W}| = N$, the time complexity in the best and worst case respectively are $\mathcal{O}(cN)$ and $\max\{\mathcal{O}(cN^3), \mathcal{O}(N^4)\} = \mathcal{O}(N^4)$.

□

Algorithm 5: GREEDY ADDING FRIENDS CS2

Input: A non-c-graph $G = (V, E)$ with a parameter c
Variables: A set $flag$ of boolean values
Output: A c-degree graph $G' = (V, E')$

1 $\mathcal{W} \leftarrow \{u \in V \mid d_u < c\}$
2 Sort the list $\mathcal{W}$ according to degree value in the increasing order
3 **foreach** $(n \in \mathcal{W})$ **do** $flag[n] \leftarrow$ false
4 **while** $\mathcal{W} \neq \varnothing$ **do**
5 $n \leftarrow$ first node in the list $\mathcal{W}$
6 **if** $(flag[n] =$ false$)$ **then**
7 $v \leftarrow$ first node in the list $\mathcal{W} \setminus \{n\}$ and not connect to n
8 **if** $(v \neq$ null$)$ **then** `ConnectUpdateInfo`(n, v)
9 **else**
10 `ConnectUpdateInfo`(n, w) where w is any normal node and $e(n, w) = 0$
11 $flag[n] \leftarrow$ true
12 **else** Connect n to arbitrary $(c - d_n)$ normal nodes w where $e(n, w) = 0$ and update d_n, d_w
13 **if** $(d_n = c)$ **then** Eliminate n from $\mathcal{W}$
14 **else** Search and place new position for n in the list $\mathcal{W}$
15 **if** $(v \neq$ null *and* $v \in \mathcal{W}$ *and* $d_v = c)$ **then**
16 Eliminate v from $\mathcal{W}$
17 **else if** $(v \neq$ null *and* $v \in \mathcal{W})$ **then**
18 Search and place new position for v in the list $\mathcal{W}$

7.2.3 Greedy algorithm

This section first introduces the greedy algorithm for our ADDFRIENDS problem and then performs computational complexity analysis.

Algorithm description

There are several greedy solutions for the ADDFRIENDS problem. Intuitively, we can apply following two strategies to choose weaker nodes and make connections:

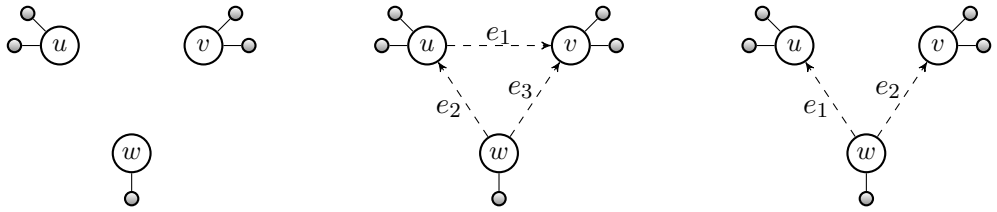
1. (**CS1**) First select one weaker node with maximum degree and then choose alternative node with maximum degree of the remaining ones.

2. (**CS2**) First select one weaker node with minimum degree and then choose alternative node with minimum degree of the remaining ones.

For the sake of simplicity, we only consider method CS2 presented in Algorithm 5 (Algorithm CS1 can be obtained from Algorithm 5 by sorting the list $\mathcal{W}$ according to degree value in the decreasing order). In Algorithm 5, we sort the (initial) list $\mathcal{W}$ only once (line 2) with the assumption that if two nodes have the same degree value then the one with smaller identity is considered to be higher priority. We try to avoid repeating sorting the ordered list $\mathcal{W}$ when one weaker node's degree is changed, because otherwise the time complexity of this algorithm becomes much higher. To achieve that complexity advantage: (i) we use a boolean set $flag$ for weaker nodes such that initially all elements are assigned *false* value, and as of time we detect that one weaker node n could not find any other weaker ones to connect, $flag[n]$ is set to *true*; (ii) if weaker node n's degree is changed and $flag[n] = false$ then we find a new position for it in the list $\mathcal{W}$ (which takes less time than sorting the list $\mathcal{W}$ again).

Greedy CS1 vs. Greedy CS2

In this part, we first define the performance comparison between two general algorithms, then analyze the comparison between CS1 and CS2 in Theorem 7.2, and finally analyze the time complexity of these greedy algorithms.



(a) Initial graph G (b) CS1 (c) CS2

Figure 7.4: Graph where CS2 is better than CS1.

Definition 7.3 (Better/Worse Algorithm)**.** *Given two algorithms A_1 and A_2. Algorithm A_1 is said to be* better (*resp.* worse) *than A_2 if there exist some graph structures G such that after deploying A_1 and A_2 into G and receiving output graphs G' and G'' respectively, we get $\Phi_{A_1}(G', G) > \Phi_{A_2}(G'', G)$ (resp. $\Phi_{A_1}(G', G) < \Phi_{A_2}(G'', G)$).*

Theorem 7.2. *There exist some graph structures such that* CS1 *is better than* CS2 *and vice versa.*

Proof. We show the claim by examining the following graph structures.

Consider the AddFriends problem for graph in Fig. 7.4a with $c = 3$.

The procedure of strategy CS1 is described in Fig. 7.4b. In this strategy, node u and v are chosen first to link (by edge e_1) and they become normal ones as $d_u = d_v = c$. The remaining node, w, could not find any weaker nodes to connect, hence, it selects any normal nodes, e.g., u and v, to become its friends by edges e_2 and e_3. Consequently, in this approach $\Phi_1 = 3$.

Alternatively, by CS2, we obtain a better value of Φ (see Fig. 7.4c). Node w with the minimum degree is selected first. As there are only two available weaker nodes u and v having same degree value, w establishes two relationships with them. After making edges e_1 and e_2, all weaker nodes u, v and w become normal ones. So we have $\Phi_2 = 2 < \Phi_1$, i.e., CS2 is better than CS1.

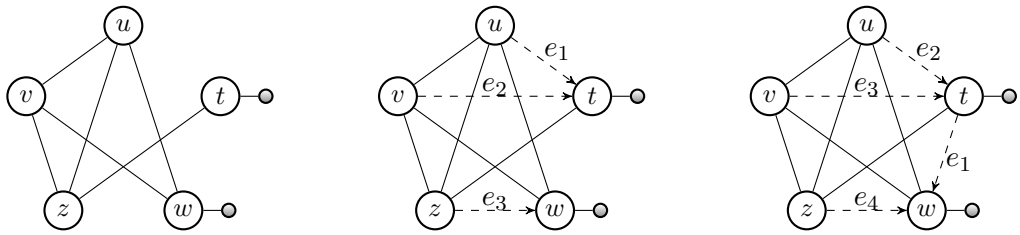
A natural question is raised in this situation: *Is strategy CS2 always better than CS1?*

The answer is no. Consider a graph in Fig. 7.5a with $c = 4$, $\mathcal{W} = \{u, v, z, w, t\}$ and $d_u = d_v = d_z = d_w = 3$, $d_t = 2$. Applying scheme CS1 into G_2 is depicted in Fig. 7.5b. As u, v, z, w are maximum degree nodes, assume firstly, we choose node u. It has only one option: making friendship e_1 with t. After this step u becomes a normal node. The remaining weaker ones have same degree of 3. Suppose we choose node v. It has to connect to t by link e_2, and then both become normal ones. Eventually, z and w establish edge e_3, and we have $\Phi_1 = 3$.

Similarly, for CS2 (see Fig. 7.5c): Because of minimum degree, node t is chosen first and linked to other minimum degree node of the remaining, for instance w, by edge e_1. Node w becomes normal. The existing weaker nodes (u, v, z, t) have same degree of 3. We select node u and t to make friendship relation e_2. Then nodes v and z have to find other normal nodes to join. For instance, v connects to t by edge e_3, and z links to w by edge e_4. Accordingly, $\Phi_2 = 4 > \Phi_1$, i.e., CS1 is better than CS2 for a graph G_2.

If we deploy algorithm AlgoCen for graphs G in Fig. 7.4a and G_2 in Fig. 7.5a, we will obtain $\Phi_{AlgoCen} = 2$ and $\Phi_{AlgoCen} = 3$ respectively. Therefore AlgoCen is always the best strategy. □

Complexity. We describe the complexities of greedy algorithms in the following Proposition 7.2. For the sake of simplicity, we only consider CS2 here.



(a) Initial graph G_2 (b) CS1 (c) CS2

Figure 7.5: Graph where CS1 is better than CS2.

Proposition 7.2 (Time Complexity)**.** *Algorithm* CS2 *takes at most* $\mathcal{O}(cN^2)$ *and at least* $\mathcal{O}(N^2)$ *in time.*

Proof. Firstly, the list of weaker nodes is sorted (w.r.t. degree value) in time $\mathcal{O}(|\mathcal{W}|\log|\mathcal{W}|)$. At the i^{th} iteration of the **while** loop with a set of weaker nodes $\mathcal{W}_i$ (where $\mathcal{W} = \mathcal{W}_1 \supseteq \mathcal{W}_2 \supseteq \ldots \supseteq \mathcal{W}_i$), we find a pair of nodes to connect by choosing the first node n in the sorted list (w.r.t. degree value) and traversing the sorted list to discover the foremost node v which is not connected to n.

In the worst case (when initially all weaker nodes are not fully connected together), this activity takes in $\mathcal{O}(|\mathcal{W}_i|)$.

In the best case (when initially all weaker nodes are fully connected together), for the first adding link, it takes $\mathcal{O}(|\mathcal{W}_i|)$ but for the remaining $(c - d_n - 1)$ adding ones, it only takes $\mathcal{O}(1)$.

Because of new degree, the position of two weaker nodes n and v in the sorted list (w.r.t. degree value) could be changed. The adjustment can be done by determining the position of their new value in the degree list and assigning the new correspondent positions in the weaker node list (using binary search in time $\mathcal{O}(\log|\mathcal{W}_i|)$) and inserting them into that place.

Each weaker node n has to link to $(c - d_n)$ nodes, thus, the number of iteration in the worst case is $\sum_{n\in\mathcal{W}}(c - d_n) = \mathcal{O}(c|\mathcal{W}|)$, and in the best case is $\mathcal{O}(|\mathcal{W}|)$.

As $\mathcal{W} = \mathcal{W}_1 \supseteq \mathcal{W}_2 \supseteq \ldots$, the time complexity in the worst case is $\mathcal{O}(|\mathcal{W}|\log|\mathcal{W}|) + \mathcal{O}((|\mathcal{W}| + \log|\mathcal{W}|).(c|\mathcal{W}|)) = \mathcal{O}(c|\mathcal{W}|^2)$, and in the best case is $\mathcal{O}(|\mathcal{W}|\log|\mathcal{W}|) + \mathcal{O}((|\mathcal{W}| + \log|\mathcal{W}|).|\mathcal{W}|) = \mathcal{O}(|\mathcal{W}|^2)$. When $|\mathcal{W}| = N$, the time complexity in the worst case and best case are respectively $\mathcal{O}(cN^2)$ and $\mathcal{O}(N^2)$. □

Greedy vs. AlgoCen

In the following part, we analyze the difference between greedy algorithms and AlgoCen in the aspect of time complexity and accuracy.

Complexity. By Propositions 7.1 and 7.2, if all weaker nodes are initially fully connected together, CS2 runs slower than AlgoCen. However in this case each weaker node has degree $\geq |\mathcal{W}| - 1$, and it requires a parameter $c \geq |\mathcal{W}|$ (since $c > d_n \geq |\mathcal{W}| - 1$). In current social networks, the average number $\overline{d}$ of friends of a user is much less than the size of network (e.g., Facebook has over one billion active users but $\overline{d} = 234$ [180]), even with a small value of c, we can obtain a value $|\mathcal{W}| \gg c$. Consequently, it is reasonable to consider $c < |\mathcal{W}|$. With that condition, it is guaranteed the subgraph only consisting of weaker nodes is not a clique, and CS2 runs faster than AlgoCen.

Approximation. We show that CS1 and CS2 are $\frac{3}{2}$-approximation algorithms in the following theorem. W.l.o.g, we only consider method CS2 here. The similar result for CS1 could be easily obtained.

Theorem 7.3. CS2 *is a* $\frac{3}{2}$*-approximation algorithm.*

Proof. Let Φ, κ and λ (resp. Φ^*, κ^* and λ^*) respectively be the structural difference, the number of added edges between two weaker nodes, and the number of added edges between one weaker node and one normal node produced by algorithm CS2 (resp. AlgoCen).

We will show that the ratio $r = \max\left(\frac{\Phi}{\Phi^*}, \frac{\Phi^*}{\Phi}\right) = \frac{\Phi}{\Phi^*} \leq \frac{3}{2}$ (since $\Phi^* \leq \Phi$) where $\Phi = \kappa + \lambda$ and $\Phi^* = \kappa^* + \lambda^*$.

By Lemma 7.1, the total added degree incident to weaker nodes is constant and equals to $\sum_{n\in\mathcal{W}} y_n = 2\kappa+\lambda = 2\kappa^*+\lambda^* = \sum_{n\in\mathcal{W}} c - d_n = c|\mathcal{W}| - \sum_{n\in\mathcal{W}} d_n$. We denote by α this constant value.

As $\kappa, \lambda \geq 0$ and $2\kappa + \lambda = \alpha$, we have $\kappa \leq \frac{\alpha}{2}$ and $\lambda \leq \alpha$. Moreover, $\kappa + \lambda = \Phi \leq 2\kappa + \lambda = \alpha \leq 2(\kappa + \lambda) = 2\Phi$, it implies $\frac{\alpha}{2} \leq \Phi \leq \alpha$.

Similarly, $\kappa^* \leq \frac{\alpha}{2}$, $\lambda^* \leq \alpha$, and $\frac{\alpha}{2} \leq \Phi^* \leq \alpha$.

It is easy to see that $\Phi = \alpha$ iff $\Phi^* = \alpha$. Indeed, suppose by applying AlgoCen into a certain graph, we obtain $\Phi^* = \alpha$. We observe that $\Phi^* = \alpha$ iff $\kappa^* = 0$ and $\lambda^* = \alpha$, i.e., by AlgoCen, each weaker node could not find other weaker one to link, and this is also true for a weaker node having minimum degree in the graph. It follows that if deploying greedy algorithm into that graph, all weaker nodes have to connect to normal ones, i.e., $\kappa = 0$ and so $\Phi = \lambda = \alpha$. Likewise, if $\Phi = \alpha$ we also get $\Phi^* = \alpha$.

To compute the maximum ratio r, we do not examine the case $\Phi = \alpha$ (and $\Phi^* = \alpha$). Instead, we should take into account the case $\frac{\alpha}{2} \leq \Phi^* < \Phi < \alpha$. In that case, the possible value of Φ and Φ^* should be $\frac{\alpha}{2} + 1 \leq \Phi \leq \alpha - 1$ and $\frac{\alpha}{2} \leq \Phi^* < \alpha$. It follows $r = \frac{\Phi}{\Phi^*} \leq \frac{\alpha-1}{\frac{\alpha}{2}} = 2 - \frac{2}{\alpha}$. Moreover, by Lemma 7.1, $\Phi > \Phi^*$ iff $\kappa^* > \kappa$. Combining with the condition $\kappa^* \leq \frac{\alpha}{2}$ and $\kappa \geq 1$ (since $\Phi < \alpha$), it infers $\frac{\alpha}{2} \geq 2$, and thus $r \leq 2 - \frac{2}{\alpha} \leq \frac{3}{2}$.

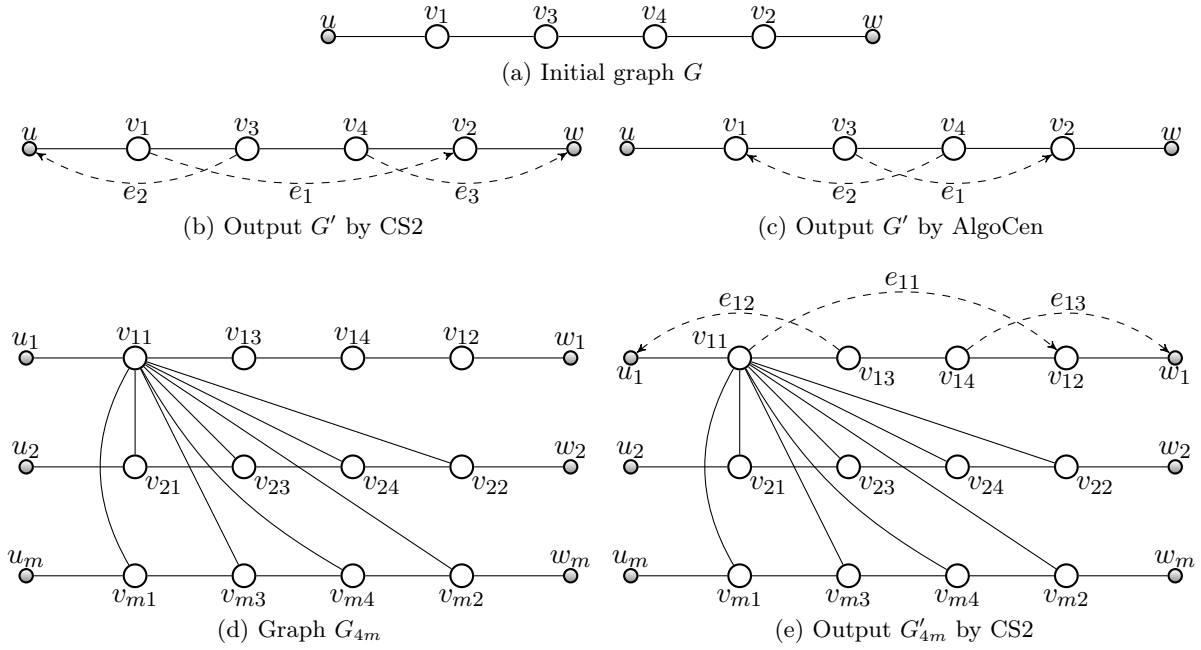
The ratio r is maximized when $\alpha = 4$, $\kappa^* = \frac{\alpha}{2} = 2$ and $\lambda^* = 0$, $\kappa = 1$ and $\lambda = 2$. This takes place when an initial graph G is the one depicted in Fig. 7.6a for $c = 3$. The output graphs G' deployed by greedy CS2 and AlgoCen are respectively presented in Fig. 7.6b and 7.6c.

More generally, we define a family of graphs G_{4m} where $m \geq 1$ (see Fig. 7.6d[16]) constructed by connecting m subgraphs G_i (same as the one described in Fig. 7.6a), such that each weaker node v_{ij} connects to other $(|\mathcal{W}|-3)$ weaker ones and one normal node (v_{ij}'s degree $d = |\mathcal{W}|-2$). For $c = d+1$, it is easy to see that by deploying AlgoCen, we obtain $\lambda^* = 0$, and $\kappa^* = \Phi^* = 2m$. In addition, by applying CS2, we get the maximum value of Φ if the output is like Fig.7.6e (that is a combination of outputs displayed in Fig. 7.6b). It implies $\kappa = m$, $\lambda = 2m$, and $\Phi = 3m$. Thus, $r = \frac{3m}{2m} = \frac{3}{2}$. It is also noted that with this family of graphs G_{4m}, we always find a graph such that the maximum difference of number of added edges between two algorithms, CS2 and AlgoCen, is equal to any given input value m. □

7.3 Decentralized protocol

So far, Section 7.2 presented an exact $\mathcal{O}(N^4)$-algorithm (in the worst case) for solving the AddFriends problem, which as far as we know is a nontrivial algorithm for this problem.

[16]For the sake of clarity, we only draw the connections from a weaker node v_{11} to other $(|\mathcal{W}|-3)$ weaker ones and one normal node.



(a) Initial graph G

(b) Output G' by CS2

(c) Output G' by AlgoCen

(d) Graph G_{4m}

(e) Output G'_{4m} by CS2

Figure 7.6: Graphs where CS2 is a $\frac{3}{2}$-approximation algorithm.

Clearly, AlgoCen is so expensive especially for social networks which are based on huge social graphs (e.g. Facebook has one billion users). That is why it is interesting to study the same problem in the decentralized model in order to get better complexity.

In this section, we investigate our AddFriends problem in decentralized model where each node only has partial knowledge over the network. We first present the system model and notations used throughout this section, then describe our distributed protocol, *AlgoDecen*, and analyze its correctness. Finally, we give the comparison between centralized and decentralized algorithms in terms of minimality of value Φ.

7.3.1 Model

In this work, we consider the $\mathcal{LOCAL}$ model (cf. [152]), a standard distributed computing model addressing to the effects of localized property. This model assumes that communication is synchronous and every node (processor) with unique identifier is woken up simultaneously, and proceeds the computation at the same round. In addition, it can exchange message of unlimited size with its neighbors as well as perform unbounded local computation.

Nodes cannot connect to arbitrary friends in the network. Instead, we assume that each node can only see and get a friend within some levels (or hops) counting from it.

Formally, we define by a constant parameter l_n, called *lookahead*, to demonstrate the highest level of neighborhood of node n. We suppose that node n recognizes the (updated) degrees of all other ones within l_n neighborhood levels. If a certain node creates a new friendship relation, this information is automatically propagated inside its scope.

For instance, if $l_n = 2$, node n can see its direct friends (1st-level) and its friends of friends (2nd-level). Any new friendship relations of n should be established between it and other nodes within these two levels of neighbors.

In this work, for the sake of simplicity, we assume all nodes have the same lookahead value l, i.e., $l_n = l$ for all n.

Algorithm 6: Decentralized Adding Friends Algorithm at node n AlgoDecen

Input:
c: constant parameter
l: lookahead of each node
$\Gamma(n)$: direct friends
$\mathcal{S}_n$: nodes in the depth up to l
$\mathcal{W}_n$: weaker nodes in the scope

Variables:
$\mathcal{D}_n$: available weaker nodes in the scope
$\mathcal{P}_n$: set of score value of weaker nodes
x_n: size of $\mathcal{D}_n$
y_n: number of edges to be added more, $y_n = c - d_n$

Output: A c-degree graph $G' = (V, E')$.

Main algorithm

1 **if** $d_n < c$ **then**
2 **while** $d_n < c$ **do**
3 `BroadcastScore()` | Event `ReceiveScore()`
4 `FindFriend()` | Event `ReceiveReq()`
5 **end**
6 **else**
7 Event `ReceiveScore()`
8 Event `ReceiveReq()`
9 **end**

Broadcasting stage

Procedure `BroadcastScore()`
10 Update $\mathcal{S}_n$, $\mathcal{W}_n$, $\mathcal{D}_n$, x_n, y_n
11 $\mathcal{P}_n[v] \leftarrow \perp$ for all $v \in \mathcal{W}_n$
12 Compute score t_n and send msg$(t_n, n, 1)$ to nodes in $\Gamma(n)$

Upon event `ReceiveScore`(t_s, s, i) **from** v **do**
13 **if** $s = n$ **then** exit
14 **if** ($d_n < c$ and $s \in \mathcal{W}_n$ and $\mathcal{P}_n[s] = \perp$) **then**
15 $\mathcal{P}_n[s] \leftarrow t_s$
16 **if** ($i < l$ and never send t_s of s) **then**
17 Forward $(t_s, s, i+1)$ to nodes in $\Gamma(n) \setminus \{v\}$

Friend Finding stage

Procedure `FindFriend()`
18 Find $v_0 \in \mathcal{S}_n$ such that: $t_{v_0} = \max\{\mathcal{P}_n\} \wedge e(n, v_0) = 0$
19 Send friendship request to v_0

Upon event `ReceiveReq()` **from** v **do**
20 **if** $d_n < c$ **then**
21 **if** ($v \in \mathcal{W}_n$ and $v = v_0$) **then**
22 Send accepted message and connect to v
23 **else** Send rejected message to v
24 **else** Send accepted message and connect to v

Furthermore, as notated in Chapter 3, for each node n, the set of its direct friends is denoted by $\Gamma(n)$ ($|\Gamma(n)| = d_n$), and the set of k-level friends of n, where $k > 1$, is recursively defined by $\Gamma_n^k = \{u \mid \delta(u, n) = k\} = \{u \mid u \in \Gamma_v \wedge v \in \Gamma_n^{k-1} \wedge u \notin \bigcup_{j<k} \Gamma_n^j\}$ where $\Gamma_n^1 = \Gamma(n)$.

We define the *scope* $\mathcal{S}_n$ of n as the set of all friends of n within distance up to l, i.e., $\mathcal{S}_n = \bigcup_{i=1}^{l} \Gamma_n^i$. To ensure there are enough neighbors to link, we assume $|\mathcal{S}_n| \geq c$.

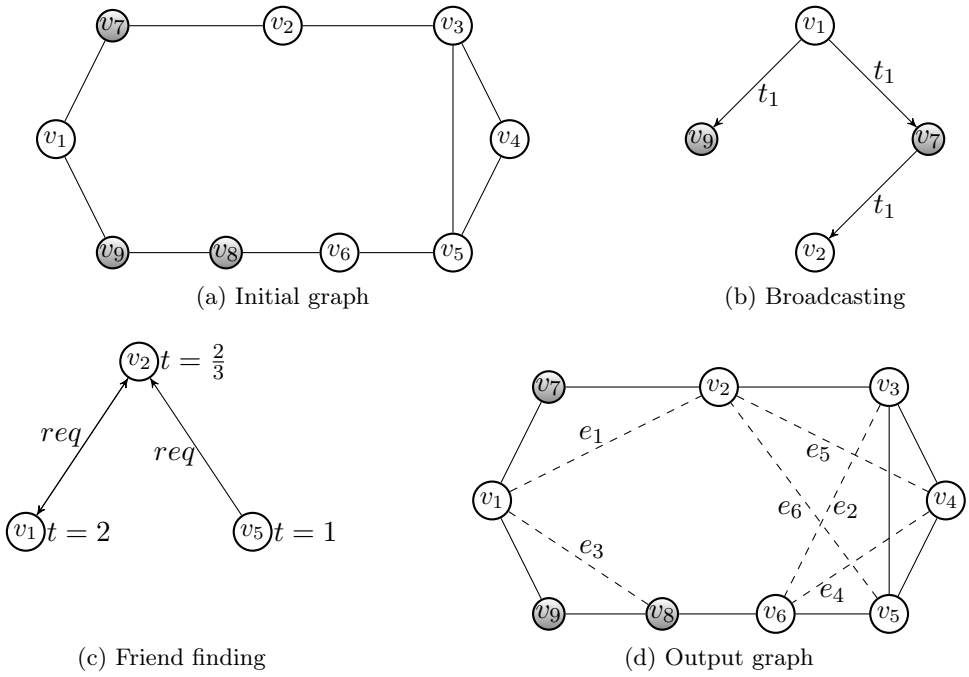
We represent $\mathcal{W}_n$ as the set of weaker nodes inside the scope of n, i.e., $\mathcal{W}_n = \{u \in \mathcal{S}_n \mid d_u < c\}$.

As mentioned in Section 7.1, adding friends is a dynamic process, and during this process, some node can join in the scope of a node. For instance, for $l = 2$, at time t_0 a node $v \in \Gamma_n^3$ and out of scope $\mathcal{S}_n$ of n but at time $t_0 + 1$ it may appear in the scope of n if a node w, such that $w \in \Gamma(v)$ and $w \in \Gamma_n^2$, requests the connection and then links n. A node may also become a normal one and leave out of the set $\mathcal{W}_n$ of n. Therefore, $\mathcal{S}_n$ and $\mathcal{W}_n$ are considered here in the dynamic process.

7.3.2 Protocol description

The decentralized algorithm, *AlgoDecen*, operates in *cycles* (see Algorithm 6). All nodes synchronize to start at the same time. At each cycle, each node has to do following two stages.

Stage 1 (Broadcasting). Each weaker node n updates its local information (lines 10–11) such as a set of weaker nodes $\mathcal{W}_n$, a set of potential friends $\mathcal{D}_n = \{v \in \mathcal{W}_n \mid e(n, v) = 0\}$ (of size x_n), the number of edges that n should add more to become a normal node $y_n = c - d_n$, and (re)initializes a set $\mathcal{P}_n$ of score values receiving from other weaker nodes in its scope. Later it computes its score t_n by using Formula (7.3), and broadcasts t_n to all direct friends (line 12). In this stage, each node also receives message $msg(t_s, s, i)$ from its direct neighbor v containing score value t_s of node s at distance $i \leq l$ (see Event ReceiveScore(t_s, s, i)). If s is a weaker node



(a) Initial graph (b) Broadcasting

(c) Friend finding (d) Output graph

Figure 7.7: Example of AlgoDecen for $c = 4$ and lookahead $l = 2$.

and inside n's scope, and this is the first updated score t_s of s which n gets, then n stores that data into $\mathcal{P}_n$ (lines 14–15) before forwarding it to other friends except v (line 17). Otherwise, it simply drops the message.
In this stage, normal nodes compute neither score nor other information. They just forward messages receiving from nodes inside their scopes to other ones.

Fig. 7.7 depicts an example for $l = 2$, $c = 4$ (initial graph is showed in Fig. 7.7a). First, each weaker node determines some information inside its scope as follows:
for v_1: $\mathcal{D}_1 = \{v_2\}$, $y_1 = 2$, $t_1 = 2$; for v_2: $\mathcal{D}_2 = \{v_1, v_4, v_5\}$, $y_2 = 2$, $t_2 = \frac{2}{3}$;
for v_3: $\mathcal{D}_3 = \{v_6\}$, $y_3 = 1$, $t_3 = 1$; for v_4: $\mathcal{D}_4 = \{v_2, v_6\}$, $y_4 = 2$, $t_4 = 1$;
for v_5: $\mathcal{D}_5 = \{v_2\}$, $y_5 = 1$, $t_5 = 1$; for v_6: $\mathcal{D}_6 = \{v_3, v_4\}$, $y_6 = 2$, $t_6 = 1$.
Fig. 7.7b shows node v_1 transmits its score t_1 to its neighbors (v_7 and v_9). This value is then forwarded by v_7 to weaker node v_2.

Once every node in the system has received score value from each of its weaker neighbors, this stage ends.

Stage 2 (Friend Finding). Each weaker node n finds one (weaker or normal) node, which currently does not link to n, and has the highest score amongst the neighbors in n's scope, to send friendship request (lines 18–19). In this stage, node n may also get friendship request from other weaker one v in its scope:

(i) If n is a weaker node: it evaluates the request by checking whether v is the one it sent offer earlier or not. In the former case, n accepts the request from v and creates a connection between them. In the later case, n refuses that request (lines 21–23).

(ii) If n is a normal node: it accepts all requests (lines 24).

Let us consider Fig. 7.7c. Node v_2 has three potential friends $\{v_1, v_4, v_5\}$ where v_1 has the highest score ($t_1 = 2$), hence, v_2 selects v_1 as its favorite choice. Node v_1 has only one available

weaker node (v_2), it sends requests to v_2. This is similar to node v_5. So node v_2 receives two requests, but it only accepts the request from v_1 as that is the node which v_2 sent request earlier. The edge e_1 is established as depicted in Fig. 7.7d. For other nodes, the similar process takes place, e.g., edge e_2 is created between v_3 and v_6.

After receiving the response from a requested node, this stage and cycle i are over. All nodes prepare information for the next cycle by updating data (e.g., degree, friendships) from its neighbors in the scope. The algorithm completes when no node is transmitting.

We reconsider an example in Fig. 7.7. By adding edges e_1 and e_2 in the first cycle, node v_3 becomes a normal one. So weaker nodes have the following information:
for v_1: $\mathcal{D}_1 = \varnothing$, $y_1 = 1$, $t_1 = 0$; for v_2: $\mathcal{D}_2 = \{v_4, v_5, v_6\}$, $y_2 = 1$, $t_2 = \frac{1}{3}$;
for v_4: $\mathcal{D}_4 = \{v_2, v_6\}$, $y_4 = 2$, $t_4 = 1$; for v_5: $\mathcal{D}_5 = \{v_2\}$, $y_5 = 1$, $t_5 = 1$;
for v_6: $\mathcal{D}_6 = \{v_2, v_4\}$, $y_6 = 1$, $t_6 = \frac{1}{2}$.

Therefore, in next cycle, v_1 requests for a connection to normal node v_8, v_2 requests to v_4, but v_4 asks for a link with v_6 which also requests to v_4, and v_5 proposes a friendship to v_2. Hence, edges e_3 (between v_1 and v_8) and e_4 (between v_4 and v_6) are generated, and we have remaining three weaker nodes v_2, v_4, v_6. We repeat this process for next cycles until there is no weaker node. The final output graph of the example is depicted in Fig. 7.7d.

7.3.3 Correctness

Lemma 7.2 (Termination)**.** *The algorithm* AlgoDecen *is ensured to eventually terminate.*

Proof. At each cycle, each node determines its favorite neighbor by requesting the node with highest score. Fig. 7.8a depicts three weaker nodes u, v, w with their corresponding scope $\mathcal{S}_u$, $\mathcal{S}_v$, $\mathcal{S}_w$, where $v \in \mathcal{S}_u$, $w \in \mathcal{S}_v$, but $w \notin \mathcal{S}_u$.

Since u (resp. v) requests friendship connection to node v (resp. w), we have $t_v = \max_{z \in \mathcal{S}_u}\{t_z\}$ (resp. $t_w = \max_{z \in \mathcal{S}_v}\{t_z\}$).

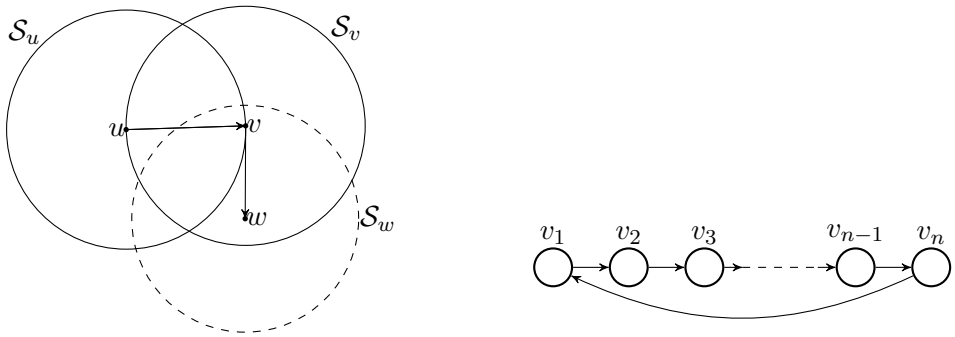
Assume, the algorithm AlgoDecen never terminates. This takes place if there exists an infinite chain: one node requests the friendship to some node, but that node also chooses other one to ask for linking, and so on. As the number of nodes is finite, that such chain must contain a loop (having at least 3 nodes). W.l.o.g., suppose we have one chain as illustrated in Fig. 7.8b. Node $v_{i+1} \in \mathcal{S}_{v_i}$ and v_i selects v_{i+1} as its favorite choice to make friendship, for all $i = 1, 2, ..., n$, where $v_{n+1} = v_1$ and $n > 2$. Because v_i chooses v_{i+1} to request connection, we have $t_{v_{i+1}} \geq t_{v_{i-1}}$ for all i, where $t_{v_{n+1}} = t_{v_1}$.

In each scope, if two nodes have the same score, then the one with the smallest identity is considered to be higher priority. Thus, w.l.o.g, all nodes in one scope have different score values. It infers $t_{v_{i+1}} > t_{v_{i-1}}$ for all i.

We consider two cases of n: (i) n is an odd number, and (ii) n is an even number. In the former case, $n = 2m + 1$ ($m \geq 1$) we have: $t_{v_2} > t_{v_{2m+1}}$, $t_{v_3} > t_{v_1}$, ..., $t_{v_{2m+1}} > t_{v_{2m-1}}$, $t_{v_1} > t_{2m}$. Hence, $t_{v_1} > t_{v_{2m}} > t_{v_{2m-2}} > ... > t_{v_2} > t_{v_{2m+1}} > t_{v_{2m-1}} > ... > t_{v_3} > t_{v_1}$. Contradiction! In the latter case, $n = 2m$ ($m > 1$) we see that $t_{v_2} > t_{v_{2m}}$, $t_{v_4} > t_{v_2}$,..., $t_{v_{2m}} > t_{v_{2m-2}}$. It implies $t_{v_2} > t_{v_{2m}} > t_{v_2}$. Contradiction! □

Corollary 7.2. *At each cycle of* AlgoDecen*, at least one connection is established.*

Proof. By Lemma 7.2, there is no loop in stage 2. It infers there exists at least one tuple of nodes (v_i, v_{i+1}) such that they request and establish connection to each other. □



(a) Scope of a node. (b) A chain with loop.

Figure 7.8: Termination property in AlgoDecen.

Proposition 7.3 (Spatial Complexity)**.** *The total space each node must hold in* AlgoDecen *is* $\mathcal{O}(N)$.

Proof. Each node n maintains a set $\mathcal{S}_n$ of neighbors (including weaker and normal ones) in its scope, a set of score values $\mathcal{P}_n$ of weaker nodes. After each cycle, the size of $\mathcal{S}_n$ can be increased, but is always not less than $|\mathcal{P}_n|$. In the worst case, all nodes in network are weaker and stored in that scope, thus the spatial complexity is $\mathcal{O}(N)$. □

Proposition 7.4 (Message Complexity)**.** *The number of messages sent by a node in* AlgoDecen *is* $\mathcal{O}(c^2N)$.

Proof. In stage 1, a node n broadcasts d_n messages to neighbors, and receives at most $|\mathcal{W}_n|$ messages comprising score values from weaker nodes. For each receiving message, it then forwards to $(d_n - 1)$ neighbors, thus there are at most $d_n + (d_n - 1)|\mathcal{W}_n|$ messages in this stage. Stage 2 requires node n to send the request to its favorite choice, as well as reply to at most $|\mathcal{W}_n|$ nodes asking for connecting. We see that one node's request is accepted, in the worst case, after $|\mathcal{W}_n|$ times of requesting. Since there are at most $(c - d_n)$ cycles (including two stages in each cycle), the number of messages each node has to send is at most $[d_n + (d_n - 1)|\mathcal{W}_n| + |\mathcal{W}_n|](c - d_n) = |\mathcal{W}_n|(d_n + 1)(c - d_n) \leq \frac{1}{4}|\mathcal{W}_n|(c + 1)^2$ (this is application of the Cauchy-Schwarz inequality). It follows the message complexity is $\mathcal{O}(c^2|\mathcal{W}_n|)$, and in the worst case where $|\mathcal{W}_n| = N$, that is $\mathcal{O}(c^2N)$. □

Proposition 7.5 (Time Complexity)**.** *Assume time evolves in rounds, i.e., each message transmission incurs a delay of at most one round. Then the protocol* AlgoDecen *operates in* $\mathcal{O}(cN(l + \log N))$ *rounds.*

Proof. Stage 1 operates $\max_n\{d_n\}.l$ rounds since it requires d_n rounds for one node n broadcasts score value to d_n direct friends which are then forwarded to weaker friends in the depth up to l. In stage 2, each weaker node finds other one to send request by first sorting the list of nodes in the scope according to score value (in time $\mathcal{O}(|\mathcal{W}_n| \log |\mathcal{W}_n|)$) and traversing the list to find the first node which is not connected to n (in time $\mathcal{O}(|\mathcal{W}_n|)$). Then the system operates l rounds for one node to send its request and at most $l|\mathcal{W}_n|$ rounds for it to response. These two stages are iterated at most $\max_n\{c - d_n\}$ cycles. Therefore, the time complexity is $\mathcal{O}((\max_n\{d_n\}l + \max_n\{|\mathcal{W}_n| \log |\mathcal{W}_n| + l + l|\mathcal{W}_n|\})(\max_n\{c - d_n\}))$. As $d_n < N$, $|\mathcal{W}_n| \leq N$, and $c - d_n < c$ for all n, in the worst case we have complexity $\mathcal{O}((lN + N \log N + l + lN)c) = \mathcal{O}(cN(l + \log N))$. □

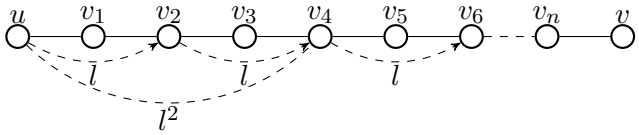


Figure 7.9: Distance $\delta(u, v)$ is reduced by a factor l.

7.3.4 Algorithms comparison

In this section, we compare the performance of different algorithms presented in this work.

Lemma 7.3. *Given a graph G and two weaker nodes u and v. The distance between u and v will be reduced at most a constant factor ($l^{2c-d_u-d_v}$) in the output graph G'.*

Proof. Assume, in the original graph G, the shortest path between u and v is $p(u, v) = \langle u, v_1, v_2, ..., v_n, v\rangle$ of distance $\delta(u, v)$ described in Fig. 7.9.

Each node can contact only with nodes in the level up to l from it. When one intermediate node connects to other node in that path, $\delta(u, v)$ will be decreased. That value is reduced at most l times when each l-th node makes a friendship with node in the l-level from it, i.e., v_l links to v_{2l}, v_{2l} links to v_{3l}, and so on. The shortest path after this adjustment is $p_1(u, v) = \langle u, v_l, v_{2l}, .., v_{\lfloor n/l \rfloor}, ..., v\rangle$, and $\delta(u, v)$ is decreased by a factor l. Similarly, if we continue that procedure, each l-th nodes in the path $p_1(u, v)$ links to other node in the l-level from them, the shortest path between u and v will be decreased more. Comparing to original path, the current path is reduced at most l^2: $p_2(u, v) = \langle u, v_{l^2}, v_{2l^2}, .., v_{\lfloor n/l^2 \rfloor}, ..., v\rangle$. As u selects $y_u = c - d_u$ nodes to connect, that above process finishes after y_u times can reduce the shortest path between u and v at most a factor l^{y_u}. For a node v, it can also make $y_v = c - d_v$ connections, distance $\delta(u, v)$ can also be reduced at most a factor l^{y_v}. It infers if u and v have to select y_u and y_v intermediate nodes respectively, their distance will be reduced at most $l^{y_u}.l^{y_v} = l^{y_u+y_v} = l^{2c-d_u-d_v}$. □

Centralized vs. Decentralized algorithms

In this part, we demonstrate the comparison between centralized and decentralized algorithms in terms of minimality of value Φ.

Theorem 7.4. *There exist some graph structures such that any deterministic decentralized algorithm is worse than a centralized algorithm.*

Proof. Let us investigate the initial graph illustrated in Fig. 7.10 for $c = 3$. In this graph, there are only two weaker nodes u and v, and the remaining ones are normal. Node u and v currently have degree $d_u = d_v = 2$, each one tries to add one more edge. The path connecting them passes normal nodes v_1 and v_2 and has a length $> l^2$. Node v_1 and v_2 are also in a subgraph G_S which diameter is greater than l^2. Because of the limitation of visibility, weaker nodes u and v do not know each other. Notice that, even we try to give the order in choosing weaker node, e.g., u is selected first, then, by Lemma 7.3 the distance between them is reduced at most a factor l, but v still does not recognize the position of u as u is outside of v's visibility. Therefore, for any decentralized algorithm, node u and v have to link to some normal nodes before generating their direct connection. It implies the total added edges for this case $\Phi_1 \geq 2$.

Furthermore, in the centralized solution, node u and v can connect to each other and we get value $\Phi_2 = 1$. This yields the desired result. □

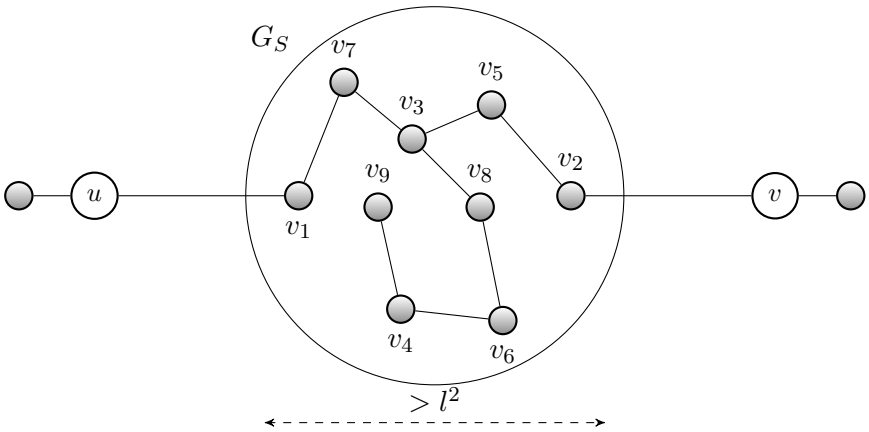


Figure 7.10: Two weaker nodes u and v cannot contact with each other.

Decentralized vs. Decentralized algorithms

We here only consider distributed algorithms which use the score value as the criterion to make decision in choosing weaker nodes. There are only three options corresponding to three methods as follows:

1. (DS1) Select weaker node with maximum score value (AlgoDecen).
2. (DS2) Select weaker node with minimum score value.
3. (DS3) Select randomly weaker node.

We now analyze the comparison amongst these decentralized algorithms as follows.

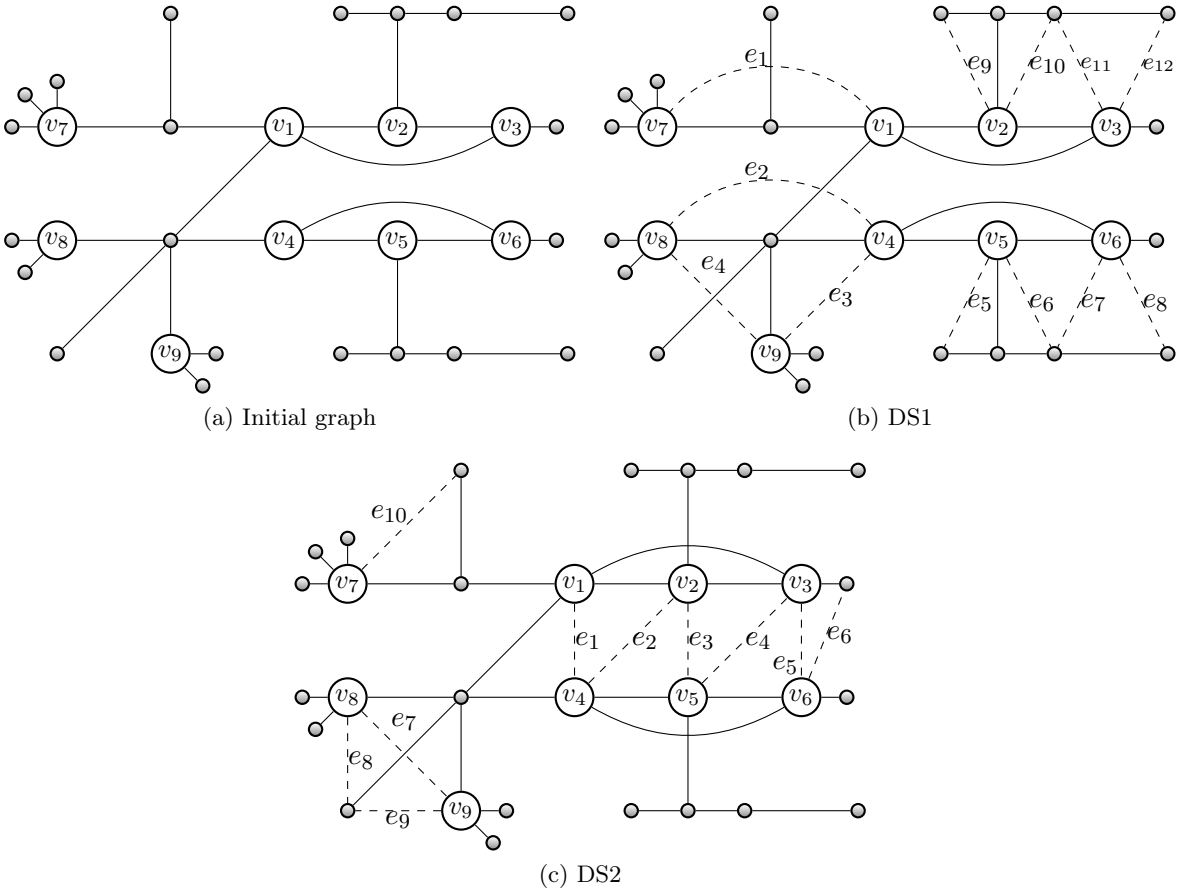
Theorem 7.5. *There exist some graph structures such that any deterministic decentralized algorithm is better than another deterministic decentralized algorithm.*

Proof. Let us consider a graph represented in Fig. 7.11 for $l = 2$ and $c = 5$. In this figure, we assume that for two nodes v_i and v_j where $i < j$, node v_i has smaller identity than node v_j, and if they have the same score value, then v_i is considered as the one having higher score than v_j.

Degree of weaker nodes are as follows: $d_{v_1} = d_{v_7} = 4$, other ones has degree 3, i.e., v_1 and v_7 have to add only one more edge, whereas other ones have to link to two nodes.

Since $l = 2$, we have sets of available nodes and their size: $\mathcal{D}_{v_1} = \{v_4, v_7, v_8, v_9\}$, $x_{v_1} = 4$; $\mathcal{D}_{v_2} = \mathcal{D}_{v_3} = \mathcal{D}_{v_5} = \mathcal{D}_{v_6} = \varnothing$, $x_{v_2} = x_{v_3} = x_{v_5} = x_{v_6} = 0$; $\mathcal{D}_{v_4} = \{v_1, v_8, v_9\}$, $x_{v_4} = 3$; $\mathcal{D}_{v_7} = \{v_1\}$, $x_{v_7} = 1$; $\mathcal{D}_{v_8} = \{v_1, v_4, v_9\}$, $x_{v_8} = 3$; $\mathcal{D}_{v_9} = \{v_1, v_4, v_8\}$, $x_{v_9} = 3$. And thus, the score value for nodes: $t_{v_1} = \frac{1}{4}$, $t_{v_7} = 1$, $t_{v_4} = t_{v_8} = t_{v_9} = \frac{2}{3}$, $t_{v_2} = t_{v_3} = t_{v_5} = t_{v_6} = 0$.

Fig. 7.11b shows us algorithm DS1: in the scope of v_1, node v_7 is the one with highest score, thus, it requests the friendship to v_7. Similarly, v_7 also asks for connection with v_1. Then the link e_1 is established, and v_1 and v_7 become normal nodes. Three nodes v_4, v_8 and v_9 have the same score but because of different identities, v_4 and v_8 request for connection to each other, whereas v_9 requires v_4 to link. Hence, v_4 accepts to be partnership with v_8 by edge e_2. Likewise, we can obtain edges e_3 and e_4 showed in Fig. 7.11b. For nodes v_2, v_3, v_5, v_6, because of the limited visibility and low score values, they could not make connection to other weaker nodes. They will link to other normal nodes by edges $e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}$ to satisfy the requirements. Consequently, we have value $\Phi_1 = 12$.



(a) Initial graph

(b) DS1

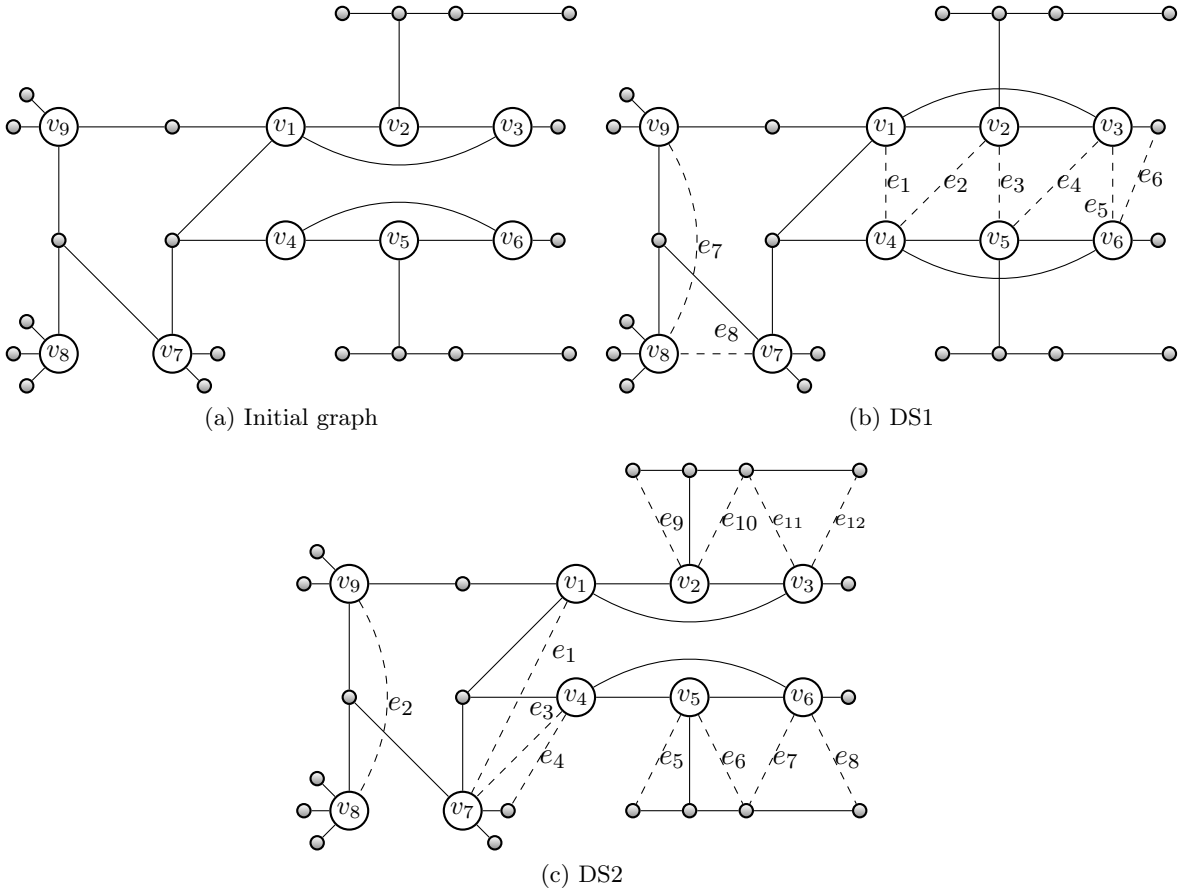
(c) DS2

Figure 7.11: Graph where DS2 is better than DS1.

Fig. 7.11c describes strategy DS2. Node v_1 (resp. v_4) is the weaker one having minimum score in the scope of v_4 (resp. v_1), and thus, they create an edge e_1. By this connection, it enables v_4 and v_2 can see each other, and hence, make a link e_2. Likewise, edges e_3, e_4, e_5, e_6 are established. Node v_8 and v_9 add edge e_7. They and node v_7 also connect to normal nodes by links e_8, e_9, e_{10} to satisfy requirements. In this case, we have $\Phi_2 = 10$. It implies, strategy DS2 is better than DS1.

However, we can obtain the reverse result, i.e., DS1 is better than DS2, in Fig. 7.12 for $l = 2$ and $c = 5$. Similarly to the analysis above, from initial graph in Fig. 7.12a, we can compute score value for nodes as follows: $t_{v_1} = t_{v_9} = \frac{1}{3}$, $t_{v_7} = \frac{1}{4}$, $t_{v_8} = \frac{1}{2}$, $t_{v_4} = 1$, $t_{v_2} = t_{v_3} = t_{v_5} = t_{v_6} = 0$. From this information, following algorithm DS1, v_1 links to v_4 by edge e_1. This enables to generate edges e_2, e_3, e_4, e_5, e_6. Continue applying DS1, with the notice that after establishing edge e_7 between v_8 and v_9, node v_7 has no available weaker nodes, hence, it connects to any normal nodes, such as v_8 by edge e_8. We show the output graph in Fig. 7.12b and $\Phi_1 = 8$. Similarly, deploying DS2 into graph in Fig. 7.12a we achieve an output graph as illustrated in Fig. 7.12c and $\Phi_2 = 12$. So for this graph, algorithm DS1 is better than DS2.

We can see through two examples, even node v_1 can choose the favorite neighbor by considering node with maximum or minimum score, but its knowledge is limited to only a part of graph, its decision could be wrong or correct for the final output graph later. Since nodes could not discover full structure of graph, its strategy can give value Φ which is not optimal. This means



(a) Initial graph

(b) DS1

(c) DS2

Figure 7.12: Graph where DS1 is better than DS2.

no deterministic strategy is the best. □

Theorem 7.6. *There exist some graph structures such that any deterministic decentralized algorithm is better than another randomized decentralized algorithm.*

Proof. Consider graph structures in Fig. 7.11a and 7.12a. In randomized decentralized algorithm, DS3, a node v_1 can only choose either v_4 or v_7 in Fig. 7.11a, v_4 or v_9 in Fig. 7.12a to make friendship. Thus the probability to obtain better value Φ is $\frac{1}{2}$. Assume we have one graph generated by combining m consecutive graphs in Fig. 7.12a and m consecutive graphs in Fig. 7.11a, where $m = \Theta(logN)$. The probability that strategy DS3 gives a minimized value Φ is $2^{-m} = O(1/N)$. When N is a large number, that probability is negligible. However, we see that by applying deterministic decentralized network such as choosing neighbor with maximum score, we can achieve a better value Φ. □

7.4 Experimental evaluation

In this section we validate our centralized solution with a performance evaluation on real-world social graphs. We start with the description of the datasets using in the experiments, then present the experimental setup, and finally show our results.

Dataset	N	$\|E\|$	CC	$\overline{d}$
DIP	20K	41K	0.52	4.1
DBLP	511K	1.9M	0.73	7.3
Youtube	1.1M	3M	0.17	5.3

Table 7.1: Datasets.

Dataset	Algorithms	Threshold c					
		31	51	71	91	121	151
DIP	AlgoCen	0.36	0.38	0.43	0.46	0.53	0.61
	CS1	0.07	0.07	0.07	0.08	0.08	0.09
	CS2	0.07	0.07	0.08	0.08	0.08	0.09
DBLP	AlgoCen	3.61	3.88	4.06	4.25	4.68	4.91
	CS1	1.78	1.85	1.89	2.09	2.25	2.46
	CS2	1.82	1.91	2.07	2.14	2.29	2.51
Youtube	AlgoCen	5.58	6.19	6.81	7.75	8.65	9.43
	CS1	3.14	3.65	3.90	4.32	4.56	4.72
	CS2	2.62	2.93	3.25	3.61	4.18	4.55

Table 7.2: Average execution time (in ms).

7.4.1 Datasets

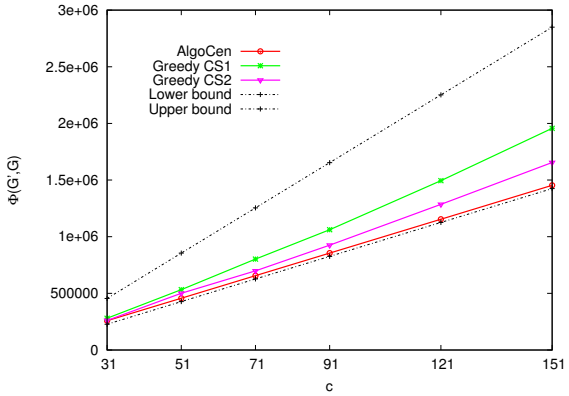
We examined our algorithm AlgoCen on the following real-world social graphs demonstrating different orders of magnitude in terms of network size.

- **DIP.** The Database of Interacting Proteins (DIP) experimentally determined interactions between proteins interpreted as undirected graph. The dataset given in [170] is used.
- **DBLP.** The DBLP dataset provides a full bibliography of publications in computer science where each node demonstrates an author, and an edge presents the relation of two authors who publish at least one paper together [121]. We use the snapshot collected in August 2008 by Sommer in [170].
- **Youtube.** Youtube is a website where users can share video clips together. The Youtube dataset includes a social graph in which each node corresponds to a user. If two users share at least one clip, they form a friendship demonstrated by an edge in the social graph. The data in our experiment is provided by Mislove *et al.* [137].

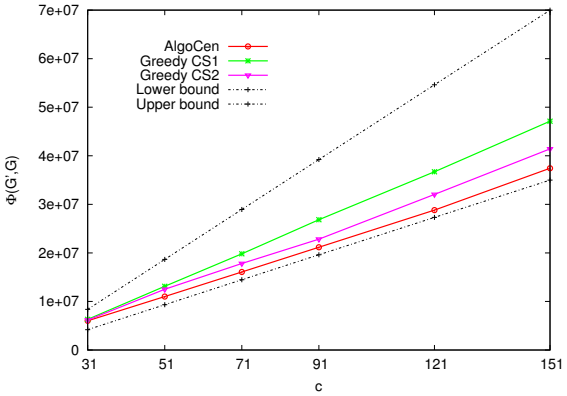
Table 7.1 summarizes some properties of datasets used in our experiments. All the graphs are unweighted, undirected and connected. For each graph, the table shows the size of network N, number of edges $|E|$, clustering coefficient CC (i.e., that is a measure of degree to which nodes in a graph tend to cluster together), and the average number of friends $\overline{d}$ for a node and $\overline{d} = 2|E|/N$.

7.4.2 Experimental setup

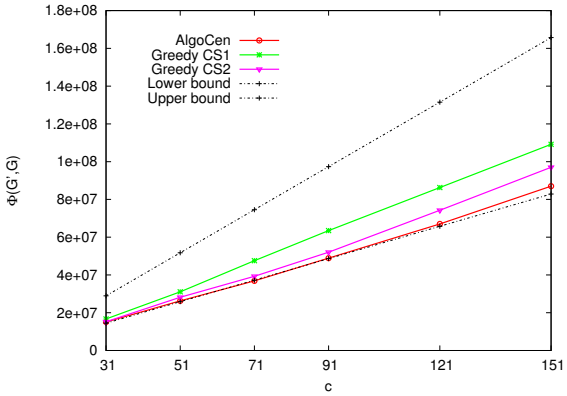
We here evaluate the practical performance of our adding friend algorithms by applying them into social graphs that are used for performing polling protocols showed in previous chapters



(a) DIP



(b) DBLP



(c) Youtube

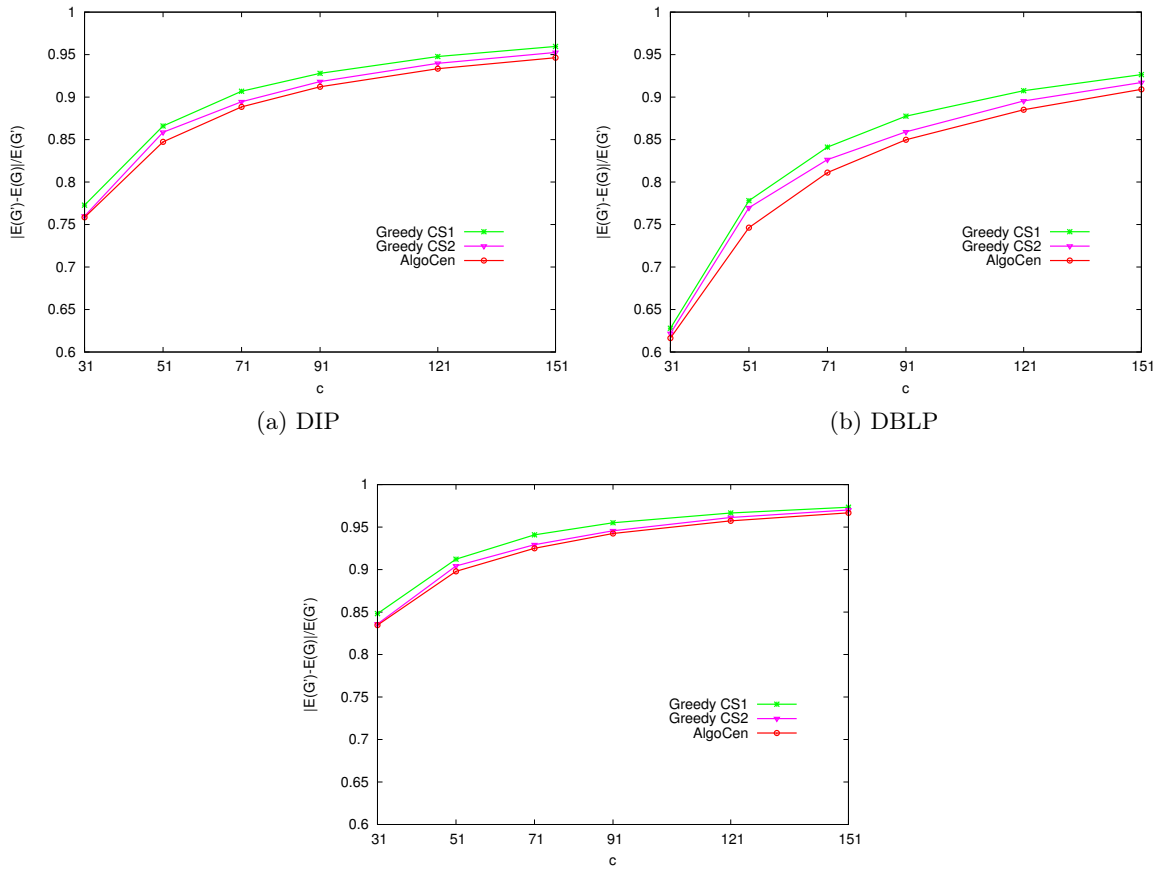
Figure 7.13: Number of added edges.

and other works like [79, 82, 83]. In other words, given a value k, we try to transform a simple general graph into other one such that a minimum degree of the output should not be less than a given threshold $c = 2k + 1$. We do the test with different values of threshold c (i.e., different privacy parameter k) and for each run we measure the *Percentage of added edges* and *Time*. The *Percentage of added edges* is computed by the formula $(|E'| - |E|)/|E|$ where E' and E are respectively the set of edges of output and input graphs. *Time* refers to the average time required to create a connection between two nodes. We do 10 independent runs for each experiment and compute the average of these experimental values.

We implement our protocols in Java and conduct the experiments on a server with 12×Quadcore AMD 64bit 1.7GHz processor, 48GB of RAM and running Ubuntu OS 12.04.

7.4.3 Results

Percentage of added edges. The number of added edges measurements by different methods are summarized in Fig. 7.13. In each figure, value Φ is represented as a function of $c = 31, 51, 71, 91, 121, 151$ (corresponding to $k = 15, 25, 35, 45, 60, 75$ in [24, 25, 65, 76, 82, 83, 95]). By Lemma 7.1 we have $\Phi = \kappa + \lambda = \frac{1}{2}\left(\sum_{n\in\mathcal{W}} y_n + \lambda\right)$. Since $0 \leq \lambda \leq \frac{1}{2}\sum_{n\in\mathcal{W}} y_n$, it implies $\frac{1}{2}\sum_{n\in\mathcal{W}} y_n \leq \Phi \leq \sum_{n\in\mathcal{W}} y_n$. Additionally, $y_n = c - \overline{d}$ for all n, we have $\frac{1}{2}\sum_{n\in\mathcal{W}}(c - \overline{d}) =$



(a) DIP

(b) DBLP

(c) Youtube

Figure 7.14: Percentage of added edges in the output graph.

$\frac{1}{2}|\mathcal{W}|(c-\overline{d}) \leq \Phi \leq \sum_{n\in\mathcal{W}}(c-\overline{d}) = |\mathcal{W}|(c-\overline{d})$. In the plots of Fig. 7.13, we observe that value Φ is always within the theoretical bounds (two black dotted lines in Fig. 7.13). In addition, algorithm AlgoCen always produces the lowest number of added edges. The curve presenting AlgoCen nearly touches the lower bound. For two greedy strategies, the results show us that CS2 is better than CS1.

We also illustrate the percentage of added edges in the output graph in Fig. 7.14, that is $\frac{E(G')-E(G)}{E(G)}$. The lower the value is, the better the structure of the input graph is preserved. We observe that AlgoCen gives the smallest percentage values.

Time. Table 7.2 describes the average execution time for generating one new edge between two nodes. This is computed as the total time required for processing the algorithm divided by value Φ. This also includes the time spent to load the input graph from disk. We see that AlgoCen is 2–6 times slower than CS1 and CS2, but the execution times still remain in few milliseconds even for the large graph like Youtube. CS1 runs slower than CS2 for the Youtube graph, but faster than CS2 in other graphs.

7.5 Conclusion

In this chapter, we formally defined the adding friends, AddFriends, problem and proposed simple and efficient algorithms for solving it in centralized and decentralized networks. Our centralized algorithm produces the optimal solution. We showed that no distributed algorithm is better than a centralized algorithm, and there is no best decentralized solution. i.e., any decentralized algorithm can be worse than other one for some graphs, but it can be better in other certain scenarios. We also conducted a set of experiments testing our proposed approaches on real-world social graphs that demonstrated that our protocols are accurate. To our best knowledge, our work is the very first theoretical study of the adding friend problem in centralized and decentralized models, using only a simple edge-addition operation.

Chapter 8

Conclusion

Contents

8.1 Summary 127
8.2 Discussion and future perspectives 129

In this chapter, we first give the summary of this thesis and then highlight our contributions to polling problem and graph transformation problem. Finally we identify the shortcomings in our work, and discuss future work.

8.1 Summary

In this thesis, we have tackled the following problems in social networks.

Polling

First we introduced the polling problem, a simple but important one for incorporating user's opinion online. The goal in studying this problem is to devise a polling protocol such that it can perform a secure and accurate process to sum up the initial users' votes with the presence of dishonest users, who try to bias the outcome and disclose the votes of honest ones. Recent works proposed polling protocols based on simple secret sharing scheme and without requiring any central authority or cryptography system [24, 25, 65, 82, 83]. However these protocols can be deployed safely and efficiently provided that, inter alia, the social graph structure should be transformed into a ring structure-based overlay and the number of participating users is perfect square. Therefore, one of the key challenges we faced throughout this thesis is to devise efficient and decentralized polling protocols that does not rely on cryptography and use more general social networks.

To address this probolem, we first surveyed existing approaches of the literature, then highlighted the gap that these protocols are too strict and solve partly the requirements of the problem. Inspired from a model of faulty nodes in distributed systems which incorporates the human and social nature of participants through privacy and reputation concerns, given in [82, 83], we proposed three designs of distributed polling protocols. For each protocol we described a family of appropriate graphs and showed their structures constitute necessary and sufficient conditions to ensure vote privacy and limit the impact of dishonest users on the accuracy of the polling output. More precisely:

Synchronous polling protocol. In Chapter 4, we described a protocol that operates in *synchronous* communication model. As we know, in a synchronous network, operations are co-

ordinated under the global control of a clock signal, and all delays of connection are limited. Furthermore, the verification procedures are provided to dissuade the dishonest misbehaviors. The verification procedures require the information about the shortest path lengths.

Asynchronous polling protocol. This is an enhanced version of the first polling protocol and has been presented in Chapter 5. More specifically, this protocol runs in the *asynchronous* network model in which the system contains no global clock; instead, it operates under distributed control. It is more difficult for nodes to predict the time of message arrival. And nodes cannot deduce the ordering of several computational events when considering the ordering of message arrivals. To overcome this trouble, we suggested a broadcasting phase where each node must send additional acknowledgments for the received data to its neighbors. Another advantage of this method is to help nodes doing the verification procedures, which prevent dishonest nodes' misbehaviors, but without requiring any extra topological knowledge such as shortest path lengths.

Polling protocol with efficient communication. In Chapter 6, we proposed an asynchronous polling protocol which does not require any verification procedures and contains a method for efficiently broadcasting message by using the concept that we call the *m-broadcasting property*. Correspondingly, instead of accepting all messages originating from a source, a node stores only m ones passed by *ordered paths*. Despite the use of richer social graph structures, the communication and spatial complexities of this protocol are close to be linear. Moreover, as messages may be corrupted by intermediate dishonest nodes, an honest node may receive distinct values emitted by a certain source. This protocol ensures each node can decide the most represented value, which is the form of majority, to obtain the correct data of other nodes. In addition, we introduced some uncertain vote disclosing rules and computed the probabilities these events occur. Furthermore, in Chapter 6, we also analyzed the affect of node crashes and message losses on accuracy and termination of the protocol by considering the impact on the final outcome and the probability of a node failing to decide and compute the final result.

Graph transformation

In Chapter 7, we addressed to the adding friends problem related to some secret sharing based protocols which impose a threshold parameter on the node degree. For instance, the minimum degree of the social graph should be not smaller than the given threshold and not all social graphs fulfill that condition. Thus, our concern is to propose a method that adds a minimum number of edges to a given graph while satisfying a threshold parameter. More formally, given a graph G and parameter c, asks for the graph satisfying the threshold c that results from G with the minimum of edge-addition operations. The problem seems simple and trivial as a special case of matching and graph anonymization problems. However, by analyzing the previous works in these fields, we showed it is not that case.

For this problem, we first proposed a centralized algorithm and proved that it is optimal solution, i.e., it produces the minimum number of added edges satisfying our requirements. Moreover, we suggested some greedy algorithms and argued that they are $\frac{3}{2}$-approximation algorithms.

As for decentralized social networks, we described some decentralized algorithms for solving adding friends problem. However, we showed that the centralized algorithm is always better than any distributed algorithms with respect to the minimum added edges. In addition, no best decentralized solution exists for this kind of problem. In other words, any decentralized algorithm, that can be better than other decentralized one for some graphs, can also be worse in some other graphs.

Finally, we tried to validate each of our solutions by implementing it and proceed a performance evaluation. All experimental results show that our protocols are accurate and inside the theoretical bounds.

8.2 Discussion and future perspectives

Beyond contributions, our work also has several limitations. In this section, we discuss these drawbacks for problems we studied and present some directions for future work.

Polling problem

First we did not validate our distributed polling protocols in practical applications. The simulation environments used in our thesis [4, 6] have several limitations compared to real world settings. For example, the simulation world assumes the existence of reliable communication among nodes, i.e., no message loss, node crashes exist. However, in the real network, these problems often take place. For future work, we plan to implement our suggested polling algorithms as a plug-in over a distributed P2P social networks such as Diaspora [1], Friendica [3], Tent [5].

Second, in the polling problem, we use the same model of adversaries as the one proposed in [82, 83]. In this model, all users care about their reputation: information related to a user is intimately considered to reflect the associated real person. Thus, they do not want their votes to be disclosed nor their misbehaviors, if any, to be publicly exposed. In particular, the dishonest users are rather restricted but more reflective of real human behavior than Byzantine ones. The dishonest users never do any misbehavior which will jeopardize their reputation and do not wrongfully blame the honest ones as this may eventually be detected, thus tarnishing their reputation. To realize this model in practice, it requires the design of an algorithm such that each user could evaluate and quantify the reputation of others by crosschecking information such as tags and social friendship relations [83].

Third, we here simply considered a binary polling with only two options in a vote. We plan to generalize our solutions to deal with *multi-options vote* in the sense a user must determine his/her favorite preference amongst multi options. Another problem that would also be interesting to study in the future is the polling with *multi-questions*. Our current work studied a polling problem with only one question, i.e., users are requested to reply for a question, then they receive the outcome that is related to the aggregation of all users votes for that question. If they are requested with more than one question, based on the results of all questions, the problem is to fulfill some statistical computations on the final results without revealing any user votes (e.g., how many users voting "Yes" for the first question).

Graph transformation problem

We plan to implement and evaluate the performance of our suggested distributed algorithms, and give comparison between them and centralized ones (including optimal and greedy protocols). That evaluation must be realized in distributed P2P social networks like Diaspora [1], Friendica [3], Tent [5].

Moreover, we also plan to consider a generalization of AddFriends problem where a graph consists of honest and dishonest nodes. For a given honest node, it completely complies with the protocol and takes care about its privacy in the sense creating friendship relations with more dishonest friends gives higher probability of being affected in other secret sharing based protocols (e.g., polling protocol). Hence, it may not randomly request connections or accept friend requests from any other nodes. The dishonest node tries to link to honest weaker ones to get as much

their information as possible. This research direction gives us a perfect novel for adding friends problem in a secure way.

Finally, we also consider the way to construct a graph that satisfies the m-broadcasting property in the future research.

Appendix A

Résumé de la Thèse en Français

Contents

A.1 Contexte 131
A.2 Motivation 132
A.3 Contributions 136
A.4 Contenu principal 138

A.1 Contexte

Les réseaux sociaux en ligne (OSNs pour Online Social Networks en anglais) est une technologie ayant pris une ampleur considérable ces dernières années dans les médias sociaux. Tout le monde peut discuter autour d'un intérêt commun, échanger des photos et des nouvelles personnelles, etc. Le nombre des utilisateurs de ces réseaux explose exponentiellement. Pour montrer un exemple typique, jusqu'à présent Facebook a plus de 1,32 milliard d'utilisateurs mensuels actifs et 829 millions d'utilisateurs quotidiens actifs en moyenne.[17] OSN permet aux participants de faire une variété d'activités liées aux affaires, au divertissement, aux événements du monde et de la culture comme le tissage de liens d'amitié, l'édition et le partage d'informations, l'échange de documents, l'expression des opinions politiques.

Un des thèmes pratiques, mais hautement sensibles, est le *problème de sondage*.[18] En général, un sondage consiste à déterminer un choix favori parmi certaines options. Chaque participant peut distribuer sa préférence en soumettant un vote, et après l'agrégation des votes, l'option majoritaire est choisie comme résultat final. Par exemple, une entreprise de téléphone mobile vient de lancer un nouveau produit et peut demander à ses clients si ses caractéristiques sont confortables, et l'utilisateur choisira l'option "Oui" ou "Non". Un autre exemple de sondage est la planification de réunion qui se compose d'une liste des options de temps où certaines tâches ou évènements sont destinés à avoir lieu, et chaque utilisateur doit exprimer son choix parmi ces options.

Une enquête en ligne a montré que le sondage en général et la planification de réunion en particulier ont un impact important dans la vie des internautes.[19] Selon cette enquête, il y a six rendez-vous en moyenne par individu chaque semaine. Le temps passé à planifier ces rendez-vous

[17] http://newsroom.fb.com/company-info/
[18] Nous utilisons ces termes "sondage", "vote" et "scrutin" désormais interchangeable.
[19] http://en.blog.doodle.com/doodle-analytics/

est de 17 minutes. En conséquence, près de 4 jours pleins sont consacrés tous les ans à coordonner des rendez-vous. En se basant sur cette étude, la planification en ligne est plus fréquemment utilisée par les internautes qui ont principalement des rendez-vous professionnels et ceux qui organisent régulièrement des rendez-vous avec au moins deux participants. Les chiffres montrent également que la planification de rendez-vous en ligne est utilisée par deux tiers des internautes Suisses et un cinquième des internautes du reste du monde.

Doodle, le plus répandu et abouti des outils pour la planification d'événements, a plus de 20 millions d'utilisateurs chaque mois dans le monde. En 2013, plus de 17 millions de sondages ont été créés, et un sondage Doodle est effectué toutes les 2 secondes. Même un petit groupe peut gagner jusqu'à 15 minutes par sondage, c'est-à-dire plus de 4 millions d'heures sont gagnées par les utilisateurs de Doodle. En 2011, plus de 30K réunions professionnelles ou privées sont organisées à travers le monde avec Doodle. Cela représente 20 nouveaux sondages Doodle lancés par minute.

Les systèmes de vote électroniques, en particulier les systèmes de vote par Internet, ont gagné en popularité et ont été utilisés pour les élections du gouvernement et les référendums au Royaume-Uni, en Estonie et en Suisse ainsi que les élections municipales au Canada et les élections primaires des partis aux États-Unis et en France. Ces systèmes permettent de faciliter les processus de prise de décision, d'accroître la participation et, dans certains cas, d'améliorer la qualité de la décision finale. Par ailleurs, au cours des dernières années, un certain nombre de tendances en ligne ont bien influencé le résultat des élections. Les médias sociaux comme OSNs peuvent augmenter la participation et provoquer des débats. En 2008, l'élection présidentielle aux États-Unis est considérée comme l'élection des médias sociaux parce que Barack Obama a beaucoup fait campagne en ligne y compris grâce à OSNs.

A.2 Motivation

Protocole de sondage. La partie principale que nous abordons dans cette thèse est le problème de sondage dans les OSNs. Nous considérons ici simplement un vote binaire avec seulement deux options "+1" ou "−1" qui représentent pour les choix "Oui" ou "Non" (ou "D'accord"/"Pas d'accord", "Pour"/"Contre"). Le but est de mettre au point un protocole de sondage sûr et précis pour agréger tous les votes initiaux avec la présence d'utilisateurs malhonnêtes qui tentent d'influencer le résultat et divulguer les votes d'utilisateurs honnêtes.

Le problème de sondage est simple, mais il a un rôle important dans l'intégration de l'avis de l'utilisateur en ligne. Par conséquent, actuellement, il y a beaucoup d'études et de solutions de ce problème dans deux contextes, réseaux centralisés et réseaux distribués. Dans les réseaux centralisés comme Facebook, toutes les données ou calculs de l'utilisateur sur ces plates-formes sont stockés et traités par l'autorité centrale qui a pleine connaissance et contrôle du réseau. Dans le cas particulier des systèmes de vote, par exemple, Facebook Poll[20] et Doodle, un serveur central est utilisé pour recueillir les votes des utilisateurs et récapituler toutes les valeurs pour obtenir le résultat final. Cependant, cette approche souffre de pannes de serveurs et de problèmes de confidentialité: généralement un utilisateur ne veut pas que son vote soit connu d'une entité centrale, et il n'est pas garanti que le serveur n'influencera pas et ne divulguera pas les scrutins des utilisateurs. En plus, il est bien connu que les OSNs centralisés détournent les informations de l'utilisateur pour un usage commercial. Pour illustrer un exemple typique, en 2009, les nouveaux termes des services de Facebook ont imposé le privilège de rester en possession

[20] http://apps.facebook.com/opinionpolls/

perpétuelle des données personnelles, même si les utilisateurs désactivent et suppriment leurs comptes.[21]Même si finalement la politique n'a pas été appliquée, cela montre la curiosité des entreprises sur les données personnelles et particulièrement sur les informations sensibles. Pour surmonter ces inconvénients d'une autorité centrale, la décentralisation d'OSN est une alternative où les utilisateurs gardent leurs propres données et effectuent des calculs de manière distribuée sans l'existence de serveur central.

Dans cette thèse, nous nous intéressons au protocole de vote basé sur un OSN décentralisé, où la vie privée de l'utilisateur est améliorée parce que ses données ne sont pas concentrées en un point central. Chaque site d'utilisateur possède les connaissances partielles de réseau et il ne lui est généralement pas possible de connaitre tout le réseau et de recueillir les informations ou les votes des autres utilisateurs. En outre, nous ne voulons pas utiliser de cryptographie pour assurer la confidentialité ou le précision du protocole parce que [82, 83]: (i) la cryptographie utilise un calcul complexe qui a un impact sur l'évolutivité et l'utilisation en pratique des protocoles; (ii) toutes les techniques de cryptographie s'appuient sur des hypothèses qui ne sont pas prouvées et pourraient être remises en cause ultérieurement, par exemple, la difficulté de factorisation d'un grand nombre ou l'inversion de fonctions mathématiques complexes; et (iii) certains problèmes traditionnels de calcul distribué peuvent être résolus sans la cryptographie telle qu'elle est motivée dans [130, 157].

Récemment, Guerraoui *et al.* ont proposé un protocole de sondage décentralisé, DPol, basé sur la partage de secret et ne nécessitant aucune infrastructure cryptographique. Dans DPol, des participants honnêtes et malhonnêtes sont considérés. Les utilisateurs honnnêtes suivent entièrement le protocole, au contraire des utilisateurs malhonnêtes qui peuvent tricher pour promouvoir leur opinion ou divulguer les votes d'utilisateurs honnêtes. En particulier, tous les participants se préoccupent de leur *réputation*: les informations relatives à un utilisateur du réseau social en ligne reflètent directement la personne réelle associée. Peu d'utilisateurs, soucieux de leur réputation, souhaitent voir leurs profils accessibles à tous et à leurs proches en particulier, annotés de la mention "tricheur". En cela, le modèle d'utilisateurs considéré est assez limité par rapport aux utilisateurs Byzantins [120]. Les auteurs ont exploité cette caratéristique des entités d'un réseau social en ligne pour dissuader les mauvais comportements des utilisateurs, au lieu d'empêcher les tricheries à l'aide de techniques mathématiques complexes (par exemple, les protocoles cryptographiques) ou de les dissimuler (par exemple, les protocoles tolérant les utilisateurs byzantins comme BFT - Byzantine Fault-Tolerance). Pour dissuader les duperies, en parallèle du protocole de sondage, les procédures distribuées de vérification sont faites pour détecter les mauvais comportements avec une probabilité non-nulle et elles permettent aux utilisateurs honnêtes de marquer les profils des utilisateurs malhonnêtes. Par exemple, si Alice est détectée comme un utilisateur malhonnête par Bob, le profil de Alice est annoté avec une marque "Alice a été détectée avec un mauvais comportement par Bob" et le profil de Bob apparaît avec une étiquette "Bob a accusé Alice comme un utilisateur malhonnête". Les auteurs ne considèrent pas les attaques Sybil puisque ces mauvais comportements peuvent être résolus par d'autres travaux tels que SybilGuard [188] et SybilLimit [187]. Par ailleurs, ils n'examinent pas de situation où les utilisateurs malhonnêtes blâment à tort les honnêtes (y compris les deux cas: un seul utilisateur accuse un groupe d'autres participants, et un groupe d'utilisateurs qui conspirent pour blâmer un autre ensemble d'utilisateurs) ou tenter de spammer le système avec beaucoup de blâmes, car il existe des outils basés sur les relations sociales entre les utilisateurs qui permettent de faire la distinction entre les accusations légitimes et illicites. Par exemple, on peut citer les systèmes de réputation comme EigenTrust [114] et PowerTrust [193], les systèmes d'atténuation

[21]http://www.nytimes.com/2009/02/19/technology/internet/19facebook.html

de mails comme Ostra [138] et SocialFilter [169], et les systèmes de recommandation comme SumUp [177] et Digg [2].

Dans DPol, les nœuds malhonnêtes peuvent former une coalition afin d'obtenir la pleine connaissance du réseau et atteindre ces objectifs suivants sans être détecté: (i) fausser le résultat du scrutin pour promouvoir leurs votes ou changer les valeurs qu'ils ont reçus d'autres nœuds honnêtes; (ii) déduire les opinions des autres nœuds. Afin d'unifier les opinions et ne pas donner les effets de compensation, tous les nœuds malhonnêtes font une seule coalition $\mathcal{D}$ de la taille D. Cependant, ils veulent aussi protéger leurs réputations. Ils sont égoïstes dans le sens où lorsque leurs complices ont été suspectés, ils préfèrent préserver leurs réputations que couvrir leurs complices.

En raison de la présence des utilisateurs malveillants, en général, la conception typique de système de vote doit remplir les caractéristiques principales suivantes:

- *Vie privée*: Le système doit protéger le vote d'utilisateur de manière à ne pas le divulguer à d'autres. En d'autres termes, un nœud malhonnête ne pourrait pas apprendre (sauf avec une probabilité négligeable) toute information de l'exécution du protocole différente de celle qu'il pourrait obtenir à partir de sa propre entrée et sortie du protocole, et aucun moyen n'existe pour en déduire ou vérifier le vote. Cette propriété se rapporte également à la résistance à la coercition du vote par examen des résultats intermédiaires publiés avant la fin du scrutin.

- *Précision*: Le résultat de sondage doit refléter correctement l'agrégation des décisions de tous les utilisateurs. Toutefois, en raison de l'absence de techniques cryptographiques qui garantissent plus facilement l'exactitude de la sortie, nous permettons ici un certain impact des nœuds malhonnêtes à la sortie. Ainsi, le protocole est dit être précis si la différence entre la sortie et le résultat attendu est négligeable.

DPol garantit la confidentialité des votes et la précision des résultats en limitant l'impact des utilisateurs malhonnêtes. Cependant, DPol a en pratique certains inconvénients.

- Premièrement, DPol repose sur des graphes qui ont une structure de recouvrement à base d'anneau [74]. Bien qu'il soit efficace pour le coût de communication, il est introduit par dessus et en dehors du graphe social normal. Il ne prend pas en compte les liens sociaux au sens où il considère une distribution uniforme des utilisateurs en groupes. Ce n'est pas pratique parce que nous devons cibler un cas particulier en utilisant la notion de groupe au lieu de réutiliser la structure normale de graphe.

- Deuxièmement, le nombre d'utilisateurs est un carré parfait tel qu'un graphe social avec N utilisateurs est divisé en N groupes de taille $\sqrt{N}$. Ainsi le protocole ne peut pas être déployé pour d'autres graphes sociaux qui ont une taille arbitraire.

Par exemple, Fig. A.1 représente la structure d'anneau utilisés dans DPol et la connexion d'un nœud n. Les N nœuds sont regroupés en $m = \sqrt{N}$ *groupes ordonné*, de g_0 à g_{m-1}. Chaque groupe est une clique, c'est-à-dire, un nœud n dans le groupe g_i est élément d'un ensemble $\mathcal{P}_o$ de *office mates*. Étant donné un paramètre k, un nœud n est lié à un ensemble de taille fixe $\mathcal{P}_p$ de nœuds, appelés *proxies*, dans le groupe suivant ($\mathcal{P}_p \subset g_{(i+1) \bmod m}$), et un ensemble de taille fixe $\mathcal{P}_c$ de nœuds, appelé *clients*, pour lesquels n agit comme un proxy, dans le groupe précédent ($\mathcal{P}_c \subset g_{i-1 \bmod m}$) où $|\mathcal{P}_p| = |\mathcal{P}_c| = 2k + 1$. Ainsi, les groupes forment un anneau où g_0 est le successeur de g_{m-1}. Chaque nœud reçoit uniquement les messages qui sont délivrés par des nœuds dans l'ensemble $\mathcal{P}_c \cup \mathcal{P}_o$, sans considérer les autres messages.

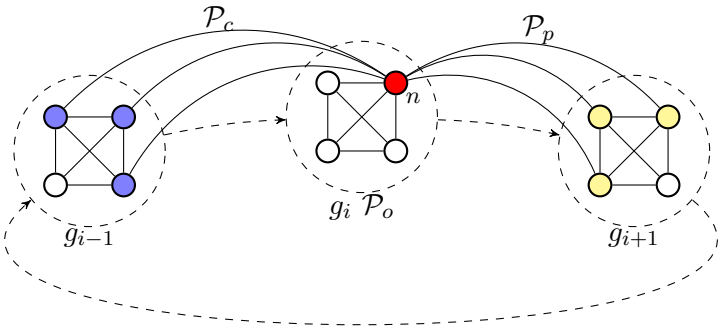


Figure A.1: Un overlay fondé sur le graphe anneau et la connexion de nœud n.

Plusieurs protocoles et extensions inspirés de DPol ont été proposés tels que MPOL [65], PDP [25] and DiPA [24]. Néanmoins, ces protocoles représentent une contribution mineure comparée à DPol, et ils reposent sur la même structure de recouvrement à base d'anneau.

De notre revue de la littérature sur le problème du scrutin, nous soulevons les questions suivantes:

- **Question 1**: *Pouvons-nous concevoir des protocoles de sondage décentralisés sans cryptographie et contraintes telles que l'utilisation de la structure de recouvrement et le nombre carré parfait d'utilisateurs?*
- **Question 2**: Une autre question, vue comme une version améliorée de la première, serait aussi intéressante: *Pouvons-nous concevoir des protocoles de sondage décentralisés avec un faible coût de communication?*
- **Question 3**: Nous sommes conscients que les sites des utilisateurs peuvent être non-sûrs ou défaillants, et les canaux de communication entre deux sites peuvent être perdus. Par conséquent, l'étude de l'effet de ces facteurs sur le protocole de sondage est également importante: *Si le réseau est non-sûr dans le sens où le site ne répond pas et les message sont perdus, comment ces facteurs influent sur la termination du protocole et la précision du résultat de sondage?*

Transformation de graphe. Un autre défi que nous étudions concerne *la transformation de graphe*. Comme introduit ci-dessus, OSNs constituent des plate-formes exploitées par un grand nombre d'utilisateurs pour effectuer des calculs à grande échelle tels que la réalisation de votes sur des questions politiques et la recherche d'informations précises sur d'énormes bases de données. Afin de préserver des données privées lors de l'excution de ces calculs, des travaux récents [65, 79, 82, 83, 181] (et les protocoles de sondage décrits dans cette thèse) utilisent un schéma de partage de secret.

Par exemple, [24, 25, 65, 76, 82, 83] ont proposé des protocoles de sondage distribués dans des réseaux sociaux ne nécessitant aucune infrastructure cryptographique. Au lieu de divulguer son vote, l'utilisateur génère un ensemble de parts qui peuvent reconstituer son vote plus tard, puis il transmet une seule part à chacun de ses voisins. Ainsi, la division du vote en parts permet aux utilisateurs de protéger la confidentialités de leur vote.

Cependant, les protocoles basés sur le partage de secret imposent un paramètre de seuil sur le nombre d'amis, par exemple, le degré minimal du graphe social ne doit pas être inférieur au

seuil donné. Malheureusement, tous les graphes sociaux ne remplissent pas cette condition. En d'autres termes, les utilisateurs ne peuvent pas effectuer de calcul élémentaire (par exemple, le processus de vote) si ce seuil n'est pas atteint. Afin de satisfaire la condition de seuil, nous essayons d'enrichir le graphe social avec de nouvelles relations d'amitié. Nous supposons que ces nouvelles relations sont acceptées par tous les utilisateur puisqu'elles sont pertinentes pour leur intérêt commun. En effet, ces relations ne sont pas contraignantes, elles n'imposent pas à leurs partenaires de partager les ressources.

Afin d'obtenir le graphe desiré, nous utilisons un ensemble d'opérations de modification de graphe appliquées sur le graphe initial. Dans cette thèse, nous ne considérons que l'opération d'adjonction de lien. Une approche naïve consiste à modifier un graphe de telle sorte que chaque nœud essaie d'ajouter autant de liens que possible pour satisfaire le seuil. Néanmoins, notre principale préoccupation dans ce travail est de répondre à la question suivante, appelée *"problème d'ajout d'amis"*: *Comment un graphe peut-il être modifié minimalement pour satisfaire le seuil sur le degré de nœud?*

Le problème d'ajout d'amis semble être simple et trivial, mais il n'est pas vraiment facile à résoudre. Même si nous essayons de résoudre ce problème dans un réseau centralisé avec la présence de serveur central qui a la connaissance globale et tout privilège sur le réseau, il n'est pas intuitif de trouver une stratégie appropriée donnant toujours la solution optimale. Ainsi, la question suivante examinée ici est d'étudier le problème d'ajout d'amis dans le réseau centralisé:

- **Question 4:** *Protocoles d'ajout d'amis avec pleine connaissance (approche centralisée).*

La résolution du problème dans le système distribué est plus difficile car il est généralement impossible pour un nœud de connaître le réseau complètement et de recueillir des informations d'autres nœuds pour demander ou accepter les relations d'amitié. Nous abordons également ce problème dans le cadre décentralisé comme suit:

- **Question 5:** *Protocoles d'ajout d'amis avec connaissance partielle (approche décentralisée).*

A.3 Contributions

Il y a cinq questions à traiter dans cette thèse. Nous y répondons en présentant les contributions clés suivantes:

1. Protocoles de sondage distribués déployés sur les réseaux sociaux originaux.
2. Protocoles d'ajout d'amis centralisé et décentralisé.

Protocoles de sondage distribués. Dans la conception de protocoles de vote distribués, nous considérons un même modèle de système que celui de [82, 83] (présenté ci-dessus).

Notre principal objectif est de garder la propriété naturelle du graphe, i.e., les utilisateurs et les liens sociaux doivent être préservés, et chaque individu peut effectuer le processus de vote en privé et en sécurité sans transformer le graphe en structure de recouvrement.

Premièrement, nous proposons la conception de protocoles de sondage décentralisés simples ne nécessitant pas une autorité centrale et un système cryptographique et utilisant un schéma de partage du secret. Deuxièmement, nous décrivons les propriétés requises sur le graphe social pour assurer l'exactitude de chaque protocole. En plus, nous couvrons une famille de graphes et montrons que leurs structures constituent des conditions nécessaires et suffisantes pour assurer la confidentialité du sondage et limiter l'impact des utilisateurs malhonnêtes sur la précision du

vote. Il est à noter qu'une structure de recouvrement à base d'anneau fait partie de notre famille de graphes.

Plus précisément, nous décrivons les trois algorithmes de sondage distribués suivants:

- Le premier protocole utilise le modèle de communication *synchrone* dans lequel tous les retards de connexion sont limités, et le système fonctionne grâce à la présence de l'horloge globale dont les impulsions doivent respecter la propriété suivante: Si un utilisateur n envoie un message à son voisin v au moment p, ce message doit arriver à v avant le moment $p+1$. Ainsi, le système s'exécute grâce à la présence d'une l'horloge globale. Mais les données envoyées par un nœud peuvent être corrompues par des nœuds intermédiaires malhonnêtes. Par conséquent, un nœud honnête peut recevoir des valeurs distinctes de la même source. Dans ce protocole, afin de prévenir les mauvais comportements des utilisateurs, nous introduisons des procédures de vérification basées sur les tables de routage et les longueurs des plus courts chemins.

- Le deuxième algorithme de vote est une version améliorée du premièr protocole. Il fonctionne dans le modèle de réseau *asynchrone* où aucune horloge globale existe et les nœuds ne peuvent pas décider leurs actions au moyen d'horloges. Un message envoyé à partir d'un nœud voisin peut arriver dans un temps fini mais imprévisible. Mais il est impossible de s'appuyer sur l'ordre d'arrivée des messages pour inférer l'ordre des calculs car l'arrivée des messages peut être arbitraire du fait des vitesses de transmission différentes. Contrairement au protocole ci-dessus, nous n'avons pas besoin de la connaissance des longueurs des plus courts chemins pour éviter les mauvais comportements.

- Malgré l'utilisation de structures de graphes sociaux plus riches (incluant la structure de recouvrement à base d'anneau présentée dans [82, 83]), un nœud peut recevoir/envoyer beaucoup de messages dupliqués d'autres/à d'autres nœuds. Cela peut conduire à une saturation du stockage local et à un coût de communication élevé. En s'inspirant de [166], nous proposons un protocole d'interrogation asynchrone qui ne nécessite pas les procédures de vérification et contient une méthode pour diffuser efficacement les messages pour une famille de graphes sociaux satisfaisant ce que nous appelons la *propriété de m-diffusion*. Un graphe satisfait la propriété de m-diffusion pour un paramètre $m \in \mathbb{N}$ tel que $1 \leq m \leq d_{min}$, où d_{min} est le degré minimal de nœud, si pour chaque nœud source, il existe un ordre topologique des nœuds tel que chaque nœud est connecté directement à la source ou à m nœuds qui le précèdent dans l'ordre par rapport à la source. En conséquence, au lieu d'accepter tous les messages émis d'une source, un nœud n'en stocke que m passés par des *chemins ordonnés*. Nous montrons que, le coût de communication et la complexité en espace de ce protocole sont proches d'être linéaires.

Pour décrire soigneusement l'implantation distribuée du problème de sondage, nous considérons les critères fondamentaux suivants: l'exactitude, la vie privée, et le nombre de nœuds malhonnêtes.

En utilisant la même notion de paramètre de confidentialité k dans [82, 83], nous obtenons les résultats suivants dans un système de taille N avec D utilisateurs malhonnêtes pour ces protocoles: (i) la probabilité que le vote d'un nœud honnête est divulgué avec certitude est au plus $(D/N)^{k+1}$; (ii) au plus $2D$ votes peuvent être révélés avec certitude par la coalition malhonnête; (iii) l'impact maximal de la coalition malhonnête sur le résultat final est $(6k+4)D$; (iv) le nombre maximal de nœuds malhonnêtes que le système peut tolérer sont respectivement $N/10$ et $(m-1)Diam(G)/2$ (où $Diam(G)$ est le diamètre du réseau) pour les deux premiers

algorithmes et la dernier. Nous validons nos solutions avec des évaluations de performance qui montrent que nos protocoles sont précis et près de l'impact moyen théorique, c'est à dire $4k + 2\alpha + 2$, où α est la proportion d'utilisateurs qui votent correctement.

Transformation de graphe. Nous décrivons d'abord le problème d'ajout d'amis qui ajoute un nombre minimum Φ de liens et satisfait un paramètre de seuil c.

Pour les réseaux sociaux centralisés, nous proposons un algorithme qui peut calculer la valeur Φ exacte avec une complexité de temps dans le pire cas $\mathcal{O}(N^4)$ (et dans le meilleur cas $\mathcal{O}(cN)$). Afin de diminuer cette borne supérieure, nous prouvons qu'il existe des algorithmes de $\frac{3}{2}$-approximation avec une complexité en temps de $\mathcal{O}(cN^2)$.

Dans les réseaux sociaux décentralisés, nous montrons qu'aucun algorithme distribué est meilleur que la solution centralisée par rapport à la valeur Φ pour toutes les structures de graphe. De plus, nous montrons également qu'il n'y a pas de meilleure solution décentralisée, c'est-à-dire, tout algorithme décentralisé peut être pire qu'un autre dans certains graphes, mais il peut être meilleur dans d'autres scénarios. Nous validons notre solution avec une évaluation de performance sur les graphes sociaux réels qui montre que nos protocoles sont exacts et dans les bornes théoriques. À notre connaissance, notre travail est la première étude théorique du problème d'ajout d'amis dans les modèles centralisé et décentralisé, en utilisant seulement l'opération d'adjonction de liens.

Publications. Certaines parties de cette thèse ont été publiées dans des conférences et soumises dans des revues:

1. B.-T. Hoang and A. Imine. On the Polling Problem for Social Networks. In *OPODIS*, pages 46–60, 2012.

2. B.-T. Hoang and A. Imine. On constrained adding friends in social networks. In *SocInfo*, pages 467–477, 2013.

3. B.-T. Hoang and A. Imine. Flexible Polling Protocol for Decentralized Social Networks. Submitted to *ACM Transactions on Internet Technology*.

A.4 Contenu principal

En bref, cette thèse est organisée comme suit.

Le chapitre 1 introduit la motivation et les contributions de la thèse.

le chapitre 2 donne un bref résumé des travaux antérieurs liés aux problèmes de sondage et la transformation de graphe. Nous présentons leurs désavantages et déterminons en quoi ces travaux se révèlent insuffisants pour satisfaire nos objectifs.

Dans le chapitre 3, nous fournissons les définitions de base et les notations qui seront utilisées dans cette thèse. Nous introduisons également les procédures de prétraitement nécessaires pour les algorithmes décrits dans cette thèse.

Dans les chapitres 4 et 5, nous présentons nos protocoles de sondage avec les procédures de vérification, respectivement pour les modèles synchrone et asynchrone de réseaux sociaux. Le chapitre 6 présente notre protocole de sondage décentralisé qui n'exige aucun processus de vérification et qui est valable pour une famille de graphes sociaux satisfaisant la propriété de m-diffusion (où m est inférieur ou égal au degré de nœud minimum). Chacun de ces chapitres commence par introduire le modèle de vote, et une famille de graphes sociaux. Ensuite, il présente le protocole de vote et analyse l'exactitude du protocole avec et sans présence de nœuds malhonnêtes. Nous validons notre solution avec une évaluation expérimentale.

De plus, les protocoles de vote dans les chapitres 4 et 5 supposent l'existence d'une communication fiable entre les nœuds. Cependant, le réseau réel peut contenir un message perdu sur

les canaux de communication ou une panne de nœuds. Dans le chapitre 6, nous analysons l'effet de ces facteurs sur la précision et la terminaison du protocole en tenant compte de l'impact sur le résultat final et la probabilité qu'un nœud ne parvient pas à décider et à calculer le résultat final.

Le chapitre 7 décrit d'abord la définition du problème d'ajout d'amis. On présente ensuite les protocoles d'ajout d'amis pour les réseaux sociaux centralisés avec leurs propriétés d'exactitude. Nous décrivons également les algorithmes distribués, en montrant qu'aucun algorithme n'est meilleur qu'un autre, et qu'ils ne peuvent pas rivaliser avec les solutions centralisées. Nous comparons les performances des protocoles en illustrant nos résultats expérimentaux.

Finalement, dans le chapitre 8, nous concluons par un résumé de cette thèse, discutons de nos réalisations ainsi que des limitations, en esquissant quelques directions de recherche pour le futur.

Bibliography

[1] Diaspora. https://www.joindiaspora.com/.

[2] Digg. http://digg.com/.

[3] Friendica. http://friendica.com/.

[4] Madkit. http://www.madkit.org/.

[5] Tent. https://tent.io/.

[6] Yalps. http://yalps.gforge.inria.fr/.

[7] M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In *EUROCRYPT*, pages 437–447, 1998.

[8] M. Abe. Mix-networks on permutation networks. In *ASIACRYPT*, pages 258–273, 1999.

[9] D. J. Abraham, A. Levavi, D. Manlove, and G. O'Malley. The stable roommates problem with globally ranked pairs. *Internet Mathematics*, 5(4):493–515, 2008.

[10] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[11] C. C. Aggarwal and P. S. Yu, editors. *Privacy-Preserving Data Mining - Models and Algorithms*, volume 34 of *Advances in Database Systems*. Springer, 2008.

[12] J. Alexander S. Kelso and V. P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50(6):1483–1504, November 1982.

[13] E. Anshelevich, O. Bhardwaj, and M. Hoefer. Friendship and stable matching. In *ESA*, pages 49–60, 2013.

[14] R. P. Anstee. A polynomial algorithm for b-matchings: An alternative approach. *Inf. Process. Lett.*, 24(3):153–157, 1987.

[15] E. Arcaute and S. Vassilvitskii. Social networks and stable matchings in the job market. In *WINE*, pages 220–231, 2009.

[16] E. M. Arkin, S. W. Bae, A. Efrat, K. Okamoto, J. S. B. Mitchell, and V. Polishchuk. Geometric stable roommates. *Inf. Process. Lett.*, 109(4):219–224, 2009.

[17] B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. *Theory Comput. Syst.*, 45(2):234–260, 2009.

[18] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

[19] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. *Commun. ACM*, 54(12):133–141, 2011.

[20] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.

[21] J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In *CRYPTO*, pages 251–260, 1986.

[22] J. C. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC*, pages 544–553, 1994.

[23] J. C. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters (extended abstract). In *PODC*, pages 52–62, 1986.

[24] Y. Benkaouz and M. Erradi. A distributed protocol for privacy preserving aggregation with non-permanent participants. *Computing*, pages 1–20, 2014.

[25] Y. Benkaouz, R. Guerraoui, M. Erradi, and F. Huc. A distributed polling with probabilistic privacy. In *SRDS*, pages 41–50, 2013.

[26] G. R. Blakley. Safeguarding Cryptographic Keys. In *AFIPS National Computer Conference*, volume 48, pages 313–317, 1979.

[27] H. L. Bodlaender, R. B. Tan, and J. van Leeuwen. Finding a bigtriangleup-regular supergraph of minimum order. *Discrete Applied Mathematics*, 131(1):3–9, 2003.

[28] P. Brandes and R. Wattenhofer. On Finding Better Friends in Social Networks. In *SSS*, pages 266–278, 2012.

[29] G. Brassard, C. Crépeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, pages 234–238, 1986.

[30] D. Bruschi, I. N. Fovino, and A. Lanzi. A protocol for anonymous and accurate e-polling. In *TCGOV*, pages 112–121, 2005.

[31] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptology*, 18(3):219–246, 2005.

[32] K. Cameron. Induced matchings. *Discrete Applied Mathematics*, 24(1-3):97–102, 1989.

[33] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[34] D. M. Cardoso, M. Kaminski, and V. V. Lozin. Maximum k -regular induced subgraphs. *J. Comb. Optim.*, 14(4):455–463, 2007.

[35] J. Casas-Roma, J. Herrera-Joancomartí, and V. Torra. An algorithm for k-degree anonymity on large networks. In *ASONAM*, pages 671–675, 2013.

[36] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[37] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[38] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[39] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.

[40] D. Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking rsa. In *EUROCRYPT*, pages 177–182, 1988.

[41] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. In *Secure Electronic Voting*, pages 211–219. 2003.

[42] D. Chaum and S. Roijakkers. Unconditionally secure digital signatures. In *CRYPTO*, pages 206–214, 1990.

[43] F. Cheah and D. G. Corneil. The complexity of regular subgraph recognition. *Discrete Applied Mathematics*, 27(1-2):59–68, 1990.

[44] B. Chen, H. Yu, Y. Zhao, and P. B. Gibbons. The cost of fault tolerance in multi-party communication complexity. *J. ACM*, 61(3):19, 2014.

[45] S. Chester, J. Gaertner, U. Stege, and S. Venkatesh. Anonymizing subsets of social networks with degree constrained subgraphs. In *ASONAM*, pages 418–422, 2012.

[46] S. Chester, B. M. Kapron, G. Ramesh, G. Srivastava, A. Thomo, and S. Venkatesh. k-anonymization of social networks by vertex addition. In *ADBIS (2)*, pages 107–116, 2011.

[47] S. Chester, B. M. Kapron, G. Srivastava, and S. Venkatesh. Complexity of social network anonymization. *Social Netw. Analys. Mining*, 3(2):151–166, 2013.

[48] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395, 1985.

[49] V. Chvátal, J. Sheehan, H. Fleischner, and C. Thomassen. Three-regular subgraphs of four-regular graphs. *Journal of Graph Theory*, 3(4):371–386, 1979.

[50] J. D. Cohen and M. J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS*, pages 372–382, 1985.

[51] G. Cornuéjols. General factors of graphs. *J. Comb. Theo.*, 45(2):185–198, 1988.

[52] C. P. Costa and J. M. Almeida. Reputation systems for fighting pollution in peer-to-peer file sharing systems. In *Peer-to-Peer Computing*, pages 53–60, 2007.

[53] R. Cramer, I. Damgård, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[54] R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung. Multi-autority secret-ballot elections with linear work. In *EUROCRYPT*, pages 72–83, 1996.

[55] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8(5):481–490, 1997.

[56] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010.

[57] I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.

[58] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, pages 119–136, 2001.

[59] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[60] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Managing and Sharing Servents' Reputations in P2P Systems. *IEEE Trans. Knowl. Data Eng.*, 15(4):840–854, 2003.

[61] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. Secretive birds: Privacy in population protocols. In *OPODIS*, pages 329–342, 2007.

[62] D. Dutta, A. Goel, R. Govindan, and H. Zhang. The design of a distributed rating scheme for peer-to-peer systems. In *P2P Econ*, 2003.

[63] F. Echenique and J. Oviedo. A Theory of Stability in Many-to-many Matching Markets. *Theoretical Economics*, 2006.

[64] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *J. Res. Nat. Bur. of Stand. Sec. B*, 69:125–130, 1965.

[65] B. Englert and R. Gheissari. Multivalued and deterministic peer-to-peer polling in social networks with reputation conscious participants. In *TrustCom/ISPA/IUCC*, pages 895–902, 2013.

[66] K. Eriksson and O. Häggström. Instability of matchings in decentralized markets with various preference structures. *Int. J. Game Theory*, 36(3-4):409–420, 2008.

[67] T. Feder, N. Megiddo, and S. A. Plotkin. A sublinear parallel algorithm for stable matching. *Theor. Comput. Sci.*, 233(1-2):297–308, 2000.

[68] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437, 1987.

[69] T. Fleiner, R. W. Irving, and D. Manlove. An algorithm for a super-stable roommates problem. *Theor. Comput. Sci.*, 412(50):7059–7065, 2011.

[70] P. Floréen, P. Kaski, V. Polishchuk, and J. Suomela. Almost Stable Matchings by Truncating the Gale-Shapley Algorithm. *Algorithmica*, 58(1):102–118, 2010.

[71] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT*, pages 244–251, 1992.

[72] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4), 2010.

[73] D. Gale and L. S. Shapley. College Admissions and the Stability of Marriage. *The Amer. Math. Month.*, 69(1):9–15, 1962.

[74] Z. Galil and M. Yung. Partitioned encryption and achieving simultaneity by partitioning. *Inf. Process. Lett.*, 26(2):81–88, 1987.

[75] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[76] S. Gambs, R. Guerraoui, H. Harkous, F. Huc, and A.-M. Kermarrec. Scalable and secure polling in dynamic distributed networks. In *SRDS*, pages 181–190, 2012.

[77] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[78] H. Ghodosi, J. Pieprzyk, and R. Steinfeld. Multi-party computation with conversion of secret sharing. *Des. Codes Cryptography*, 62(3):259–272, 2012.

[79] A. Giurgiu, R. Guerraoui, K. Huguenin, and A.-M. Kermarrec. Computing in Social Networks. *J. Infor. and Comp.*, 2013.

[80] A. Giurgiu, R. Guerraoui, K. Huguenin, and A.-M. Kermarrec. Computing in social networks. *Information and Computation*, 234(0):3–16, 2014.

[81] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[82] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, and M. Monod. Decentralized Polling with Respectable Participants. In *OPODIS*, pages 144–158, 2009.

[83] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, M. Monod, and Y. Vigfusson. Decentralized polling with respectable participants. *J. Parallel Distrib. Comput.*, 72(1):13–26, 2012.

[84] M. Gupta, P. Judge, and M. H. Ammar. A reputation system for peer-to-peer networks. In *NOSSDAV*, pages 144–152, 2003.

[85] D. Gusfield and R. W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.

[86] M. M. Halldórsson, R. W. Irving, K. Iwama, D. Manlove, S. Miyazaki, Y. Morita, and S. Scott. Approximability results for stable marriage problems with ties. *Theor. Comput. Sci.*, 306(1-3):431–447, 2003.

[87] M. M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa. Randomized approximation of the stable marriage problem. *Theor. Comput. Sci.*, 325(3):439–465, 2004.

[88] L. Harn and C. Lin. Strong (n, t, n) verifiable secret sharing scheme. *Inf. Sci.*, 180(16):3059–3064, 2010.

[89] S. Hartung, C. Hoffmann, and A. Nichterlein. Improved upper and lower bound heuristics for degree anonymization in social networks. In *SEA*, pages 376–387, 2014.

[90] S. Hartung, A. Nichterlein, R. Niedermeier, and O. Suchý. A refined complexity analysis of degree anonymization in graphs. In *ICALP (2)*, pages 594–606, 2013.

[91] O. Hasan, L. Brunie, and E. Bertino. Preserving privacy of feedback providers in decentralized reputation systems. *Computers & Security*, 31(7):816–826, 2012.

[92] O. Hasan, L. Brunie, E. Bertino, and N. Shang. A decentralized privacy preserving reputation protocol for the malicious adversarial model. *IEEE Transactions on Information Forensics and Security*, 8(6):949–962, 2013.

[93] M. Hay, G. Miklau, D. Jensen, D. F. Towsley, and C. Li. Resisting structural re-identification in anonymized social networks. *VLDB J.*, 19(6):797–823, 2010.

[94] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, pages 539–556, 2000.

[95] B.-T. Hoang and A. Imine. On the Polling Problem for Social Networks. In *OPODIS*, pages 46–60, 2012.

[96] M. Hoefer. Local Matching Dynamics in Social Networks. In *ICALP (2)*, pages 113–124, 2011.

[97] M. Hoefer. Local matching dynamics in social networks. *Inf. Comput.*, 222:20–35, 2013.

[98] S. Holzer and R. Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *PODC*, pages 355–364, 2012.

[99] R. W. Irving. An efficient algorithm for the "stable roommates" problem. *J. Algorithms*, 6(4):577–595, 1985.

[100] R. W. Irving, T. Kavitha, K. Mehlhorn, D. Michail, and K. E. Paluch. Rank-maximal matchings. *ACM Transactions on Algorithms*, 2(4):602–610, 2006.

[101] R. W. Irving and D. Manlove. The stable roommates problem with ties. *J. Algorithms*, 43(1):85–105, 2002.

[102] R. W. Irving and D. Manlove. An 8/5-approximation algorithm for a hard variant of stable marriage. In *COCOON*, pages 548–558, 2007.

[103] R. W. Irving and D. Manlove. Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. *J. Comb. Optim.*, 16(3):279–292, 2008.

[104] Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, pages 577–594, 2010.

[105] K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *ICALP*, 1999.

[106] K. Iwama, S. Miyazaki, and N. Yamauchi. A 1.875: approximation algorithm for the stable marriage problem. In *SODA*, pages 288–297, 2007.

[107] K. Iwama, S. Miyazaki, and N. Yamauchi. A (2-c(1/sqrt(n)))-approximation algorithm for the stable marriage problem. *Algorithmica*, 51(3):342–356, 2008.

[108] K. Iwama, S. Miyazaki, and H. Yanagisawa. A 25/17-approximation algorithm for the stable marriage problem with one-sided ties. *Algorithmica*, 68(3):758–775, 2014.

[109] M. Jakobsson. A practical mix. In *EUROCRYPT*, pages 448–461, 1998.

[110] M. Jakobsson. Flash mixing. In *PODC*, pages 83–89, 1999.

[111] M. Jakobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, pages 339–353, 2002.

[112] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT*, pages 143–154, 1996.

[113] R. J. B. Jr. and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

[114] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW*, pages 640–651, 2003.

[115] T. Kavitha and C. D. Shah. Efficient algorithms for weighted rank-maximal matchings and related problems. In *ISAAC*, pages 153–162, 2006.

[116] S. Khazaei and D. Wikström. Randomized partial checking revisited. In *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, pages 115–128, 2013.

[117] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994.

[118] Z. Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60(1):3–20, 2011.

[119] B. Korte and J. Vygen. Combinatorial optimization. *Alg. Comb.*, 21, 2008.

[120] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[121] M. Ley and P. Reuther. Maintaining an Online Bibliographical Database: The Problem of Data Quality. In *EGC*, 2006.

[122] Z. Li, H. Shen, and K. Sapra. Leveraging social networks to combat collusion in reputation systems for peer-to-peer networks. *IEEE Trans. Computers*, 62(9):1745–1759, 2013.

[123] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD Conference*, pages 93–106, 2008.

[124] Y.-X. Liu, L. Harn, C.-N. Yang, and Y. Zhang. Efficient (n, t, n) secret sharing schemes. *Journal of Systems and Software*, 85(6):1325–1332, 2012.

[125] L. Lovász. The factorization of graphs. In *Combinatorial Structures and Their Apps*, pages 243–246, 1970.

[126] L. Lovász. The factorization of graphs. II. *Act. Ma. Aca. Sc. Hung.*, 23:223–246, 1972.

[127] L. Lovász and M. D. Plummer. *Matching Theory.* North-Holland Math. Stud. Elsevier Science Publisher, North-Holland, 1986.

[128] X. Lu, Y. Song, and S. Bressan. Fast identity anonymization on graphs. In *DEXA (1)*, pages 281–295, 2012.

[129] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 1(1), 2007.

[130] D. Malkhi, O. Margo, and E. Pavlov. E-voting without 'cryptography'. In *Financial Cryptography*, pages 1–15, 2002.

[131] D. Malkhi and E. Pavlov. Anonymity without 'cryptography'. In *Financial Cryptography*, pages 108–126, 2001.

[132] D. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theor. Comput. Sci.*, 276(1-2):261–279, 2002.

[133] L. Mathieson and S. Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *J. Comput. Syst. Sci.*, 78(1):179–191, 2012.

[134] U. M. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[135] R. C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978.

[136] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, pages 223–228, 2004.

[137] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC*, 2007.

[138] A. Mislove, A. Post, P. Druschel, and P. K. Gummadi. Ostra: Leveraging trust to thwart unwanted communication. In *NSDI*, pages 15–30, 2008.

[139] G. D. F. Morales, A. Gionis, and M. Sozio. Social content matching in mapreduce. *PVLDB*, 4(7):460–469, 2011.

[140] O. Moreno and V. A. Zinoviev. Three-regular Subgraphs of Four-regular Graphs. *European Journal of Combinatorics*, 19(3):369–373, 1998.

[141] H. Moser and D. M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *J. Discrete Algorithms*, 7(2):181–190, 2009.

[142] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In *CRYPTO*, pages 573–590, 1999.

[143] V. Nikov and S. Nikova. On a relation between verifiable secret sharing schemes and a class of error-correcting codes. In *WCC*, pages 275–290, 2005.

[144] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *ICICS*, pages 440–444, 1997.

[145] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols Workshop*, pages 25–35, 1997.

[146] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[147] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, pages 248–259, 1993.

[148] E. Pavlov, J. S. Rosenschein, and Z. Topol. Supporting privacy in decentralized additive reputation systems. In *iTrust*, pages 108–119, 2004.

[149] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[150] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[151] T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *EUROCRYPT*, pages 522–526, 1991.

[152] D. Peleg. Distributed computing: a locality-sensitive approach. SIAM, 2000.

[153] D. Peleg, L. Roditty, and E. Tal. Distributed algorithms for network diameter and girth. In *ICALP (2)*, pages 660–672, 2012.

[154] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[155] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989.

[156] P. Resnick and R. Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. *Advances in applied microeconomics*, 11:127–157, 2002.

[157] R. L. Rivest. Chaffing and winnowing: confidentiality without encryption. In *RSA Laboratories CryptoBytes 4*, 1998.

[158] M. Rodriguez-Perez, O. Esparza, and J. L. Muñoz. Analysis of peer-to-peer distributed reputation schemes. In *CollaborateCom*, 2005.

[159] E. Ronn. Np-complete stable matching problems. *J. Algorithms*, 11(2):285–304, 1990.

[160] K. Sako. Electronic voting schemes allowing open objection to the tally. *IEICE*, E77-A(1), 1994.

[161] K. Sako and J. Kilian. Secure voting using partially compatible homomorphisms. In *CRYPTO*, pages 411–424, 1994.

[162] K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995.

[163] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng.*, 13(6):1010–1027, 2001.

[164] P. Samarati and L. Sweeney. Generalizing Data to Provide Anonymity when Disclosing Information. In *PODS*, 1998.

[165] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *CRYPTO*, pages 148–164, 1999.

[166] N. B. Shah, K. V. Rashmi, and K. Ramchandran. Secure network coding for distributed secret sharing with low communication cost. In *ISIT*, pages 2404–2408, 2013.

[167] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[168] A. Singh and L. Liu. Trustme: Anonymous management of trust relationships in decentralized p2p systems. In *Peer-to-Peer Computing*, pages 142–149, 2003.

[169] M. Sirivianos, K. Kim, and X. Yang. Socialfilter: Introducing social trust to collaborative spam mitigation. In *INFOCOM*, pages 2300–2308, 2011.

[170] C. Sommer. Dblp graph. http://www.sommer.jp/graphs/.

[171] S. Song, K. Hwang, R. Zhou, and Y.-K. Kwok. Trusted p2p transactions with fuzzy reputation aggregation. *IEEE Internet Computing*, 9(6):24–34, 2005.

[172] M. Srivatsa, L. Xiong, and L. Liu. Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks. In *WWW*, pages 422–431, 2005.

[173] I. A. Stewart. On locating cubic subgraphs in bounded-degree connected bipartite graphs. *Discrete Mathematics*, 163(1-3):319–324, 1997.

[174] L. J. Stockmeyer and V. V. Vazirani. Np-completeness of some generalizations of the maximum matching problem. *Inf. Process. Lett.*, 15(1):14–19, 1982.

[175] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[176] A. Tamir and J. S. B. Mitchell. A maximum b-matching problem arising from median location models with applications to the roommates problem. *Math. Program.*, 80:171–194, 1998.

[177] D. N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *NSDI*, pages 15–28, 2009.

[178] W. T. Tutte. A short proof of the factor problem for finite graphs. *Canad. J. Math.*, 6:347–352, 1954.

[179] W. T. Tutte. Spanning subgraphs with specified valencies. *Discrete Mathematics*, 306(10-11):932–938, 2006.

[180] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *CoRR*, 1111.4503, 2011.

[181] L.-H. Vu, K. Aberer, S. Buchegger, and A. Datta. Enabling Secure Secret Sharing in Distributed Online Social Networks. In *ACSAC*, pages 419–428, 2009.

[182] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing (awarded best paper). In *3rd Symposium on Networked Systems Design and Implementation (NSDI 2006), May 8-10, 2007, San Jose, California, USA, Proceedings.*, 2006.

[183] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans. Knowl. Data Eng.*, 16(7):843–857, 2004.

[184] M. Yamashita and T. Kameda. Computing on an anonymous network. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 117–130, 1988.

[185] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, pages 739–750, 2008.

[186] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical robust communication in dhts tolerating a byzantine adversary. In *2010 International Conference on Distributed Computing Systems, ICDCS 2010, Genova, Italy, June 21-25, 2010*, pages 263–272, 2010.

[187] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. *IEEE/ACM Trans. Netw.*, 18(3):885–898, 2010.

[188] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman. Sybilguard: defending against sybil attacks via social networks. *IEEE/ACM Trans. Netw.*, 16(3):576–589, 2008.

[189] M. Yuan, L. Chen, and P. S. Yu. Personalized privacy protection in social networks. *PVLDB*, 4(2):141–150, 2010.

[190] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *PinKDD*, pages 153–171, 2007.

[191] B. Zhou and J. Pei. Preserving Privacy in Social Networks Against Neighborhood Attacks. In *ICDE*, 2008.

[192] B. Zhou and J. Pei. The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowl. Inf. Syst.*, 28(1):47–77, 2011.

[193] R. Zhou and K. Hwang. Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Trans. Parallel Distrib. Syst.*, 18(4):460–473, 2007.

[194] R. Zhou, K. Hwang, and M. Cai. Gossiptrust for fast reputation aggregation in peer-to-peer networks. *IEEE Trans. Knowl. Data Eng.*, 20(9):1282–1295, 2008.

[195] A. Zwierko and Z. Kotulski. A light-weight e-voting system with distributed trust. *Electr. Notes Theor. Comput. Sci.*, 168:109–126, 2007.

Résumé

Un des thèmes pratiques, mais hautement sensibles, est le problème de sondage dans les réseaux sociaux où le caractère secret des informations sélectionnées et la réputation de l'utilisateur sont très critiques. En effet, les utilisateurs désirent préserver la confidentialité de leurs votes et dissimuler, le cas échéant, leurs mauvais comportements. Récemment, Guerraoui *et al.* ont proposé des protocoles de sondage basés sur la partage de secret et ne nécessitant aucune infrastructure cryptographique. Néanmoins, ces protocoles ne sont applicables que si le graphe social a une structure d'anneau et le nombre d'utilisateurs est un carré parfait. Dans cette thèse, nous traitons, d'une part, du problème du déploiement décentralisé des protocoles de sondage qui sont basés sur des graphes sociaux ayant des structures générales, et d'autre part, du problème de transformation des graphes sociaux pour augmenter les propriétés de vie privée et de précision, nécessaires au déroulement sûr et rentable du sondage décentralisé. Premièrement, nous proposons trois protocoles décentralisés qui s'appuient sur l'état originel (sans transformation) des graphes sociaux. Les deux premiers protocoles utilisent respectivement des modèles de communication synchrone et asynchrone, et manipulent des procédures de vérification pour détecter les utilisateurs malhonnêtes. Quant au troisième protocole, il est asynchrone et ne nécessite pas de procédures de vérification. Pour que ce protocole permette une diffusion efficace de messages, nous avons défini une propriété sur les graphes sociaux, appelée "m-broadcasting". Dans la deuxième partie de la thèse, nous formalisons le problème de "l'ajout des amis" qui consiste à trouver une transformation optimale des graphes sociaux pour les adapter au partage de secret. Pour résoudre ce problème, nous présentons deux algorithmes selon deux approches différentes: centralisée et décentralisée. Une évaluation expérimentale montre que nos protocoles sont précis et restreints aux bornes théoriques.

Mots-clés: Réseaux sociaux, Protocole de sondage, Partage de secret, Vie privée, Ajout de contacts, Problème d'édition dans les graphes, Algorithmes centralisé et décentralisé.

Abstract

One of the current practical, useful but sensitive topic in social networks is polling problem where the privacy of exchanged information and user reputation are very critical. Indeed, users want to preserve the confidentiality of their votes and to hide, if any, their misbehaviors. Recently, Guerraoui *et al.* proposed polling protocols based on simple secret sharing scheme and without requiring any central authority or cryptography system. However these protocols can be deployed safely and efficiently provided that the social graph structure should be transformed into a ring structure-based overlay and the number of participating users is perfect square. In this thesis, we address the problem of deploying decentralized polling protocols for general social graphs and how to transform these graphs in order to increase the privacy and/or accuracy properties. First, we propose three simple decentralized polling protocols that rely on the current state of social graphs. The two first protocols use synchronous and asynchronous models and verification procedures to detect the misbehaving users. The third protocol is an asynchronous one that does not require any verification procedures and contains a method for efficiently broadcasting message under a family of social graphs satisfying what we call the m-broadcasting property. Second, we formalize the "adding friends" problem such that we can reuse the social graphs after some minimum structural modifications consisting in adding new friendship relations. We also devise algorithms for solving this problem in centralized and decentralized networks. We validate our solutions with some performance evaluations which show that our protocols are accurate, and inside the theoretical bounds.

Keywords: Social networks, Polling protocol, Secret sharing, Privacy, Adding friends, Graph editing, Centralized and decentralized algorithms.